A Scalable Blocking Framework for Multidatabase Privacy-preserving Record Linkage

Thilina Ranbaduge

A thesis submitted for the degree of
Doctor of Philosophy in Computer Science
The Australian National University

February 2018

Except where otherwise indicated, this thesis is my own original work.

Thilina Ranbaduge
Friday 16th February, 2018

I dedicate this thesis to my beloved family for their unconditional love, care, and support through out my life.

Acknowledgments

I would like to express my sincere appreciation to all those who have contributed to this thesis and supported me in one way or the other during this amazing journey.

Firstly, I would like to express my sincere gratitude to my primary supervisor, Peter Christen, for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. My Ph.D has been an amazing experience and I thank Peter wholeheartedly, not only for his tremendous academic support, but also for giving me so many wonderful opportunities. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having the best supervisor and mentor for my Ph.D study.

Similar, profound gratitude goes to my second supervisor, Dinusha Vatsalan, who offered me an invaluable assistance, support and guidance throughout this Ph.D research. Dinusha is a knowledgeable, dedicated and motivating supervisor and I have learned greatly from her. Also, I would like to thank my third supervisor, Vassilios S. Verykios, who has been very supportive and helpful in providing constructive feedback on my research with his expertise. I would also like to extend my thanks to Rainer Schnell and Sean Randall, whom I have co-authored with, for their valuable contributions.

I am sincerely thankful for the funding provided by the Australian Research Council (ARC Discovery Project DP130101801) for my studies, without which it would not have been possible. I also thank the Research School of Computer Science and the Australian National University for offering me an opportunity to conduct my research studies. The school and university are well supportive of students.

I thank my fellow colleagues which I have been fortunate to work with. For the stimulating discussions, feedback, support, and encouragement, I like to thank Jeffrey Fisher, Qing Wang, Yichen Hu, Jingyu Shao, and Anushka Isuru.

Last but not least, I would like to thank my family for all their love and encouragement. For my parents, Karunadasa and Sunethra, who raised me with lots of love and still support me in all my pursuits. Also a special thank should go to my sister Nilini for being an inspiration and providing guidance throughout my life. Also, I would like to thank my in-laws, Parakrama and Mavis, for providing continuous help and support through out this Ph.D journey.

Finally, I would like to show my gratitude to two important persons in my life my wife, Ishara, and my son, Akitha. Both of you have been a constant source of strength and inspiration for me. There were times during the past years when everything seemed hopeless and I did not have any hope. Your determination and constant encouragement ultimately made it possible for me to see this Ph.D through to the end. Thank you everybody.

“Imagination is more important than knowledge. For knowledge is limited to all we now know and understand, while imagination embraces the entire world, and all there ever will be to know and understand.”
-Albert Einstein

Abstract

Today many application domains, such as national statistics, healthcare, business analytic, fraud detection, and national security, require data to be integrated from multiple databases. Record linkage (RL) is a process used in data integration which links multiple databases to identify matching records that belong to the same entity. RL enriches the usefulness of data by removing duplicates, errors, and inconsistencies which improves the effectiveness of decision making in data analytic applications.

Often, organisations are not willing or authorised to share the sensitive information in their databases with any other party due to privacy and confidentiality regulations. The linkage of databases of different organisations is an emerging research area known as privacy-preserving record linkage (PPRL). PPRL facilitates the linkage of databases by ensuring the privacy of the entities in these databases.

In multidatabase (MD) context, PPRL is significantly challenged by the intrinsic exponential growth in the number of potential record pair comparisons. Such linkage often requires significant time and computational resources to produce the resulting matching sets of records. Due to increased risk of collusion, preserving the privacy of the data is more problematic with an increase of number of parties involved in the linkage process.

Blocking is commonly used to scale the linkage of large databases. The aim of blocking is to remove those record pairs that correspond to non-matches (refer to different entities). Many techniques have been proposed for RL and PPRL for blocking two databases. However, many of these techniques are not suitable for blocking multiple databases. This creates a need to develop blocking technique for the multidatabase linkage context as real-world applications increasingly require more than two databases.

This thesis is the first to conduct extensive research on blocking for multidatabase privacy-preserved record linkage (MD-PPRL). We consider several research problems in blocking of MD-PPRL. First, we start with a broad background literature on PPRL. This allow us to identify the main research gaps that need to be investigated in MD-PPRL. Second, we introduce a blocking framework for MD-PPRL which provides more flexibility and control to database owners in the block generation process. Third, we propose different techniques that are used in our framework for (1) blocking of multiple databases, (2) identifying blocks that need to be compared across subgroups of these databases, and (3) filtering redundant record pair comparisons by the efficient scheduling of block comparisons to improve the scalability of MD-PPRL. Each of these techniques covers an important aspect of blocking in real-world MD-PPRL applications. Finally, this thesis reports on an extensive evaluation of the combined application of these methods with real datasets, which illustrates that they outperform existing approaches in term of scalability, accuracy, and privacy.

List of Publications

- **Publications contribute to this thesis:**

  1. **A Scalable and Efficient Subgroup Blocking Scheme for Multidatabase Record Linkage:** Thilina Ranbaduge, Dinusha Vatsalan, and Peter Christen. In proceedings of the 22nd Pacific-Asia knowledge Discovery and Data Mining Conference (PAKDD), Melbourne, Australia, June 2018. (Full paper, Tier A Conference, Acceptance rate 17.2%) - corresponds to Chapter 8 of this thesis.
  2. **Scalable Block Scheduling for Efficient Multi-Database Record Linkage:** Thilina Ranbaduge, Dinusha Vatsalan, and Peter Christen. In proceedings of the 16th IEEE International Conference on Data Mining (ICDM), Barcelona, Spain, December 2016. (Short paper, Tier A* Conference, Acceptance rate 19.6%) - corresponds to Chapter 9 of this thesis.
  3. **Hashing-based Distributed Multi-party Blocking for Privacy-preserving Record Linkage:** Thilina Ranbaduge, Dinusha Vatsalan, Peter Christen, and Vassilios Verykios. In proceedings of the 20th Pacific-Asia knowledge Discovery and Data Mining Conference (PAKDD), Auckland, New Zealand, April 2016. In Advances in Knowledge Discovery and Data Mining, Volume 9652, Pages 415-427. (Full paper, Tier A Conference, Acceptance rate 17.2%) - corresponds to Chapter 7 of this thesis.
  4. **Clustering-Based Scalable Indexing for Multi-Party Privacy-Preserving Record Linkage:** Thilina Ranbaduge, Dinusha Vatsalan, and Peter Christen. In proceedings of the 19th Pacific-Asia knowledge Discovery and Data Mining Conference (PAKDD), Ho Chi Minh City, Vietnam, May 2015. In Advances in Knowledge Discovery and Data Mining, Volume 9078, Pages 549-561. (Full paper, Tier A Conference, Acceptance rate 15.6%) - corresponds to Chapter 6 of this thesis.
  5. **Tree Based Scalable Indexing for Multi-Party Privacy-Preserving Record Linkage:** Thilina Ranbaduge, Dinusha Vatsalan, and Peter Christen. In proceedings of the 12th Australasian Data Mining Conference (AusDM), Brisbane, Australia, November 2014. In CRPIT, Volume 158, Pages 31-42. (Full paper, Tier B Conference, Acceptance rate 45.6%) - corresponds to Chapter 5 of this thesis.

- **Other publications:**

  1. **Pattern Mining based Cryptanalysis of Bloom Filters for Privacy-Preserving Record Linkage:** Peter Christen, Anushka Vidanage, Thilina Ranbaduge, Rainer Schnell. In proceedings of the 22nd Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Melbourne, Australia, June 2018. (Full paper, Tier A Conference, Acceptance rate 10%)
  2. **Efficient Cryptanalysis of Bloom Filters for Privacy-Preserving Record Linkage:** Peter Christen, Rainer Schnell, Dinusha Vatsalan, Thilina Ranbaduge. In proceedings of the 21st Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Jeju, South Korea, May 2017. In Advances in Knowledge Discovery and Data Mining, Volume 10234, Pages 628-640. (Full paper, Tier A Conference, Acceptance rate 28.2%)
  3. **Evaluation of Advanced Techniques for Multi-Party Privacy-Preserving Record Linkage on Real-World Health Databases:** Thilina Ranbaduge, Dinusha Vatsalan, Sean Randall, and Peter Christen. In proceedings of the International Population Data Linkage Conference (IPDLN), Swansea, Wales, August 2016. In International Journal of Population Data Science (IJPDS), Volume 1, Number 1.
  4. **MERLIN - A Tool for Multi-party Privacy-preserving Record Linkage:** Thilina Ranbaduge, Dinusha Vatsalan, and Peter Christen. In proceedings of the 15th IEEE International Conference on Data Mining Workshop (ICDMW), Atlantic City, USA, November 2015. (Demo paper, Tier A* Conference)

Contents

List of Figures

List of Tables

Notation and Terminology

Notation

$\mathbf{B}$	A set of blocks
$\mathbf{BF}$	Inverted index of Bloom filters
$\mathbf{BDP}$	List of block description pairs
$\mathbf{CBT}$	List of candidate block tuples
$\mathbf{D}$	A database
$\mathbf{G}$	A graph data structure
A	Set of blocking attributes
B, B_{rep}	A block of records and block representative
BDP	A block description pair
CBT	Candidate block tuple
CRT	Candidate record tuple
DR	Disclosure risk value
E	List of edges
$\mathbb{G}$	Global (publicly available) database
$\mathcal{H}$	Set of hash functions
PS	Probability of suspicion value
Q	An queue data structure
S	A set of q-grams
$R, R.a, R.id$	A record, record attribute value, and record identifier
V	A list of vertices
V_B, V_R	A binary and a ratio vector
bf	Bloom filter

b_{min}, b_{max}	Minimum and maximum block size
d	Number of database owners (or databases)
fp	False positive rate
$h(\cdot)$	A hash function used for mapping a value into a Bloom Filter
n_h, n_q, n_r	Number of hash functions, q-grams, and records
l_q, l_{bf}	Length of a q-gram and a Bloom filter
$sim()$	A similarity function
q	A q-gram
w, w_t	Weight of an edge and weight threshold

Terminology

BF	Bloom filter
BDP	Block description pair
BK	Blocking key
BKV	Blocking key value
BRP	Block representative pair
DO	Database owner
FM	F-measure
LSH	Locality sensitive hashing
LU	Linkage unit
MD-PPRL	Multidatabase Privacy-preserving record linkage
MDRL	Multidatabase record linkage
MHS	Min-Hash signature
PC	Pairs completeness
PPRL	Privacy-preserving record linkage
QID	A quasi-identifier
RL	Record linkage
RR	Reduction ratio
SMC	Secure multi-party computation

Chapter 1

Introduction

In this chapter we provide an introduction to the work presented in this thesis. We describe multidatabase privacy-preserving record linkage (MD-PPRL) and the application areas of multidatabase linkage in Section 1.1. We then describe the research problem addressed by this thesis in Section 1.2, the aim of the research study in Section 1.3, our contributions towards the research problem in Section 1.4, and the methodology addressing the research problem in Section 1.5. Lastly, we provide the organisation of the thesis in Section 1.6.

1.1 Multidatabase Privacy-preserving Record Linkage

Many organisations, including businesses, government agencies and research organisations, are collecting vast amounts of data into their databases. Such data is stored, processed, and analysed for interesting patterns and knowledge that support efficient and quality decision making for the improvement of the organisations' works [70, 121, 217]. These databases often contain many thousands or even millions of records. The size of databases is continuously increasing, and developing techniques for efficient processing, analysing and mining has gained much recognition in both academia and industry [22, 24, 74, 151, 191].

To improve the efficiency and effectiveness in decision making and to conduct sophisticated data analysis it is often required the databases from different organisations to be integrated [179]. Data integration not only enriches data, but it can also improve the quality of data by identifying duplicate or similar records that refer to the same real-world entity [66, 96, 155]. A real-world entity represented by a record in a database can be a person, a product, a business, or any other object that exists in the real world. The process of matching and aggregating records that relate to the same entity from different data sources is known as *record linkage*, *data matching* or *entity resolution* [66, 71, 96]. For the rest of this thesis we will use the term record linkage (RL). RL is often performed on databases where records are about people [35].

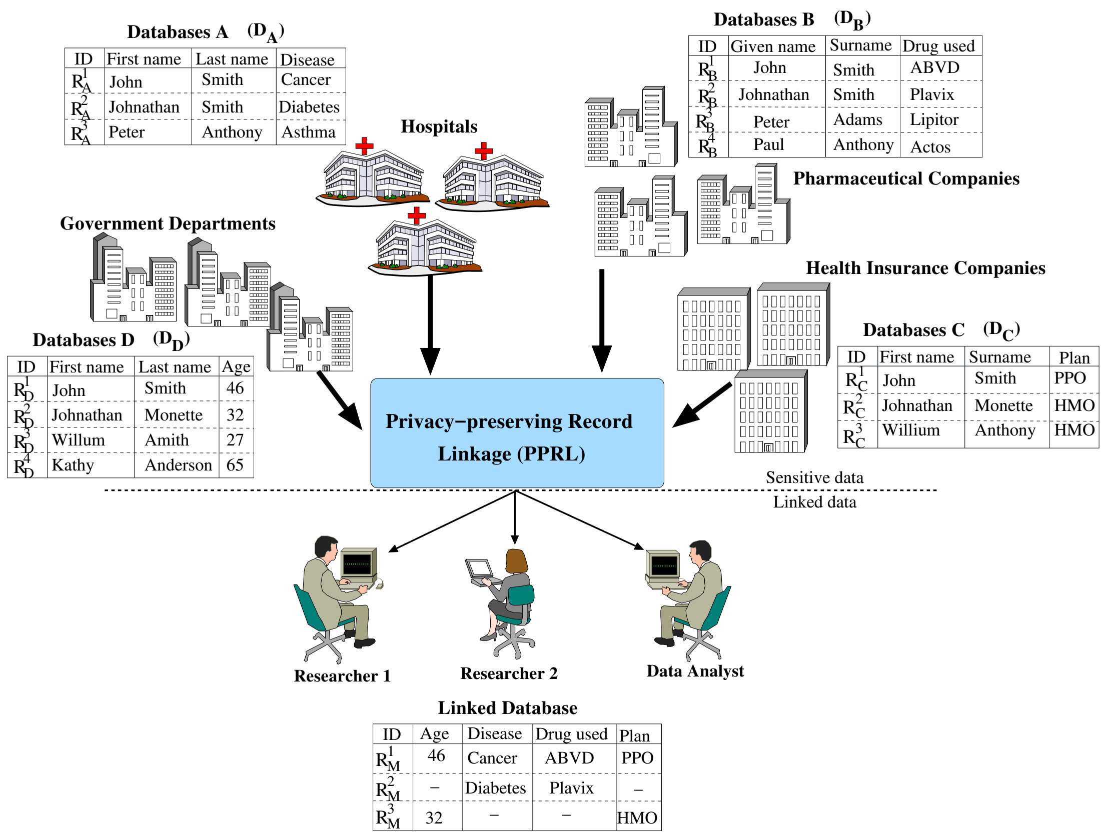
In most situations the linkage of records from multiple sources requires large computational resources for processing. The linkage process becomes even more challenging when records do not contain unique entity identifiers across all the databases that need to be linked. Though personal information contained in those records (such as first name, last name, address details, age, etc.) could be used for linking records, organisations commonly do not want such sensitive information to be revealed to other data sources due to growing privacy and confidentiality concerns [177, 189, 216].

The emerging research area to find records in multiple data sources that relate to the same entity without revealing personal information is known as *privacy-preserving record linkage*, *blind data linkage* or *private record linkage* [3, 42, 88, 189, 218]. For the rest of this thesis we will use the term privacy-preserving record linkage (PPRL). The record pairs or sets that refer to the same entity are known as *matches* and the record pairs or sets that refer to different entities are known as *non-matches* [35]. Figure 1.1 illustrates an application scenario of PPRL.

Any linkage application that performs the task of identifying and matching records that refer to the same entities from multiple databases faces various challenges. Scalability, linkage quality, and privacy are the three key challenges that are associated with PPRL [176, 212].

1. **Scalability:** The Big Data era generates major concerns in record linkage applications as many of the linkage techniques employed in these applications are not scalable to very large databases [36, 66, 216, 217]. Scalability generally depends on the complexity of the linkage techniques employed as they can create bottlenecks in the whole linkage process. In two database linkage, as a naïve approach, each record in one database can be compared with every other record in other database to identify the matches. The quadratic complexity consists in the naïve pair-wise comparison of records across two databases which does not scale when the numbers of records are increasing in each database. As we discuss in Section 1.2, this becomes even more problematic when the number of databases to be linked is increasing as the naïve pair-wise comparison of records grows exponentially. For example, in a linkage of three databases each record pair resulted in the linkage of first two databases needs to be compared with each record in a third database. Multidatabase linkage, the task of linking more than two databases, requires more sophisticated and efficient ways to reduce the record comparison space to make the linkage process more scalable to larger databases as well as an increasing number of databases.

2. **Linkage Quality:** The classification of record sets from multiple databases into matches and non-matches is the main aim of record linkage [38]. Such classification is performed based on the similarity vectors generated by the comparison functions (such as edit distance, Jaccard, Jaro, and Jaro-Winkler string similarity functions [35]) applied on different attributes of record pairs [35, 58]. On occasions where unique identifiers for entities are available across all the databases to be linked, a simple database join can be used to identify the matching pairs of records. However, in most cases such a common identifier is not available in all databases to be linked.



Databases A (D_A)

ID	First name	Last name	Disease
R_A^1	John	Smith	Cancer
R_A^2	Johnathan	Smith	Diabetes
R_A^3	Peter	Anthony	Asthma

Databases B (D_B)

ID	Given name	Surname	Drug used
R_B^1	John	Smith	ABVD
R_B^2	Johnathan	Smith	Plavix
R_B^3	Peter	Adams	Lipitor
R_B^4	Paul	Anthony	Actos

Databases C (D_C)

ID	First name	Surname	Plan
R_C^1	John	Smith	PPO
R_C^2	Johnathan	Monette	HMO
R_C^3	Willium	Anthony	HMO

Databases D (D_D)

ID	First name	Last name	Age
R_D^1	John	Smith	46
R_D^2	Johnathan	Monette	32
R_D^3	Willum	Amith	27
R_D^4	Kathy	Anderson	65

Linked Database

ID	Age	Disease	Drug used	Plan
R_M^1	46	Cancer	ABVD	PPO
R_M^2	–	Diabetes	Plavix	–
R_M^3	32	–	–	HMO

Figure 1.1: An example health surveillance system that uses PPRL to link different databases. In this example, the researchers and a data analyst (also known as data consumer) aim to investigate correlations between illnesses and different drugs and how people pay for their treatments. This requires data from government departments, hospitals, pharmaceutical companies, and health insurance companies to be linked. Once the linkage is performed the data consumers only receive an aggregated result (selected attribute values that cannot be used to identify personal information) of the matching record sets.

A possibility to overcome this issue is to use quasi-identifiers (QID) such as first name, last name, address details, age, and so on [95]. The QIDs used in record comparisons can produce ambiguous results as real-world data can contain errors, variations, missing values, and values that can be out of date [174]. Therefore, exact matching of QID values is not sufficient to achieve accurate linkage between databases, and approximate matching techniques are generally required to achieve high linkage quality in RL applications [96, 220].

3. **Privacy:** In PPRL, the use of QIDs that contain personal information for linking different databases poses the challenge of protecting the privacy of the data or the privacy of the entities, to whom the data relates to, being used for the linkage [177, 219]. For example, information such as medical or financial details

of individuals cannot be revealed or communicated unless such information is appropriately de-identified (anonymised) or encrypted when linking data from multiple databases [29]. When personal information is used for linking databases across organisations, privacy of this information needs to be considered because when linked, detailed information about individuals that is even more revealing might become available. For example, de-duplicated information can be matched with publicly available information to re-identify the individual to which the data belongs to. To avoid such re-identification process, it is required that the data used for linking across multiple databases and the identities of entities represented by data, as well as the sensitive details of the matching results of the linkage, are privacy preserved throughout the record linkage process [44, 216].

As shown in Figure 1.1, PPRL only outputs an aggregated result to the data consumers that does not contain any personal or sensitive information.

Given these three main challenges, any PPRL application should be capable of: (1) handling large numbers of records from multiple databases, (2) correctly identifying the true matching records to achieve high linkage quality, and (3) preserving the privacy of sensitive attribute values of all records. We formally define multidatabase PPRL (MD-PPRL) as follows [216]:

Definition 1.1. Multidatabase privacy-preserving record linkage (MD-PPRL)
Assume DO_A, DO_B, $\cdots$, DO_X are d database owners with their respective databases $\mathbf{D}_A$, $\mathbf{D}_B$, $\cdots$, $\mathbf{D}_X$. The linkage across all these databases determines which of their records ${R_A}^i \in \mathbf{D}_A$, ${R_B}^j \in \mathbf{D}_B, \cdots$, ${R_X}^k \in \mathbf{D}_X$ match according to a decision model $\mathcal{C}(\langle {R_A}^i, {R_B}^j, \cdots, {R_X}^k \rangle)$, that accepts a tuple of records $\langle {R_A}^i, {R_B}^j, \cdots, {R_X}^k \rangle$ as input, where $1 \leq i \leq |\mathbf{D}_A|$, $1 \leq j \leq |\mathbf{D}_B|$, and $1 \leq k \leq |\mathbf{D}_X|$, and $|\mathbf{D}|$ represents the number of records in a database $\mathbf{D}$. $\mathcal{C}(\langle {R_A}^i, {R_B}^j, \cdots, {R_X}^k \rangle)$ assigns each $\langle {R_A}^i, {R_B}^j, \cdots, {R_X}^k \rangle$ into the two sets $\mathbf{M}$ of matches (where records ${R_A}^i$, ${R_B}^j, \cdots$, and ${R_X}^k$ are assumed to refer to the same entity, i.e. $\mathbf{M} = \{\langle {R_A}^i, {R_B}^j, \cdots, {R_X}^k \rangle | {R_A}^i = {R_B}^j = \cdots = {R_X}^k\}$) and $\mathbf{U}$ of non-matches (where the records refer to different entities, i.e. $\mathbf{U} = \{\langle {R_A}^i, {R_B}^j, \cdots, {R_X}^k \rangle | {R_A}^i \neq {R_B}^j \neq \cdots \neq {R_X}^k\}$). Each such tuple contains records from two or more databases. The objective of MD-PPRL is that at the end of the linkage process the database owners learn only which records they have in common according to $\mathcal{C}$ without revealing the actual values of the records R^i_A, $R^j_B, \cdots, R^k_X$ with any other database owner or any party external to the database owners.

1.1.1 Applications of Multidatabase Linkage

Many application domains require information from multiple sources to be integrated and combined in order to improve data quality and to facilitate further analysis of data [22, 66]. The following are some example application scenarios that illustrate why multidatabase record linkage (MDRL) is important and valuable.

1.1.1.1 Census

As a first example, assume a government census agency collects data about various aspects of the population and economy of a country. The information collected is used to generate a diverse range of statistical reports for planning the allocation of funding and resources by the government [77]. Generally, each census is collected at a different point in time and likely into a different database with same schemas (assuming same information is collected about individuals in each different point in time), allowing multiple such databases to be integrated to compile new statistical information. Each database commonly contains longitudinal information about individuals, such as age, gender, birthplace, birth year, religion, race, highest educational qualification, and so on[1][2]. It is commonly accepted that such data are important sources of information to help identify the characteristics of a population as it changes over time [41, 131, 188]. This potentially requires the linkage of such large census databases collected over several decades [78].

1.1.1.2 Health Analytics

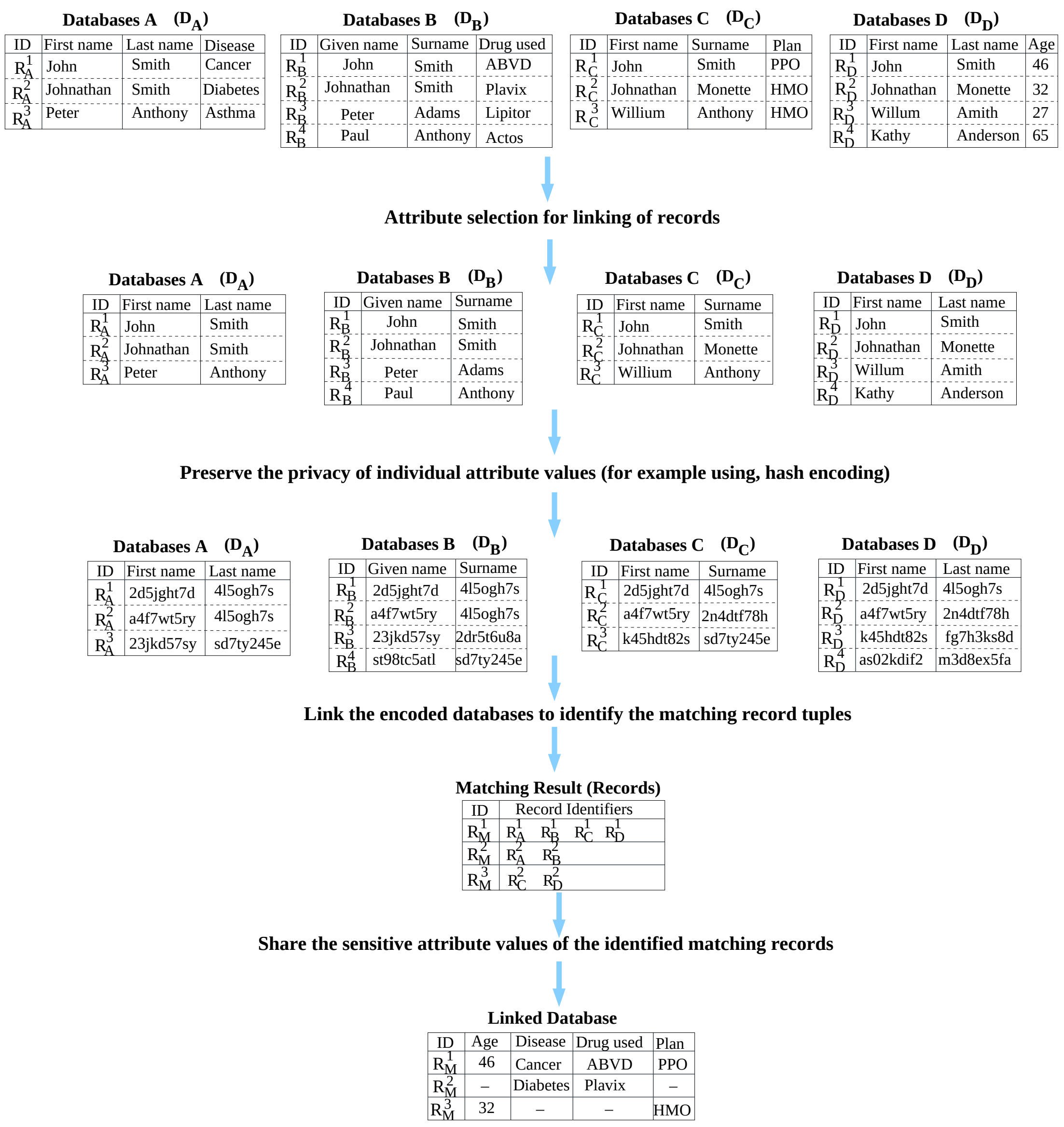
Another real-world example application, as shown in Figure 1.1, would be a health surveillance system that continuously integrates and links data from hospitals, pharmacies, insurance companies, and government departments [22, 29]. The data collected from these sources can help to investigate the geographical and temporal effects of diseases and drug usages in certain patient groups. Such a system requires the collection and linkage of different databases, as illustrated in Figure 1.2, from certain regions of a country, each potentially containing many hundreds of thousands of patient records, where each record needs to be linked across all the different databases to identify how patient groups with different illnesses react to different drugs and medical treatments [24].

As illustrated in Figures 1.1 and 1.2, the databases from different organisations often contain different schemas which require a health surveillance system to select possible attributes that are common in all databases for linking. Next, the values of these attributes are encoded (masked) to preserve privacy. The linkage of records is then performed by comparing the encoded values, and the resulting sets of matching record identifiers are sent to the database owners. Finally, the surveillance system generates a linked database that contains the required sets of attribute values of these matched records that can be used for making decisions [24, 29, 57].

1.1.1.3 National Security and Crime Investigation

Another real-world scenario is an counter-terrorism application[3][4] to track known terrorists or criminal suspects within a country [72, 180]. These applications are gen-

[1]http://www.abs.gov.au/websitedbs/censushome.nsf/home/2016?opendocument&navpos=110
[2]https://www.census.gov/
[3]https://www.globalsecurity.org/security/systems/tia.htm
[4]https://www.dhs.gov/sites/default/files/publications/privacy-matrix-122006.pdf



Databases A (D_A)

ID	First name	Last name	Disease
R_A^1	John	Smith	Cancer
R_A^2	Johnathan	Smith	Diabetes
R_A^3	Peter	Anthony	Asthma

Databases B (D_B)

ID	Given name	Surname	Drug used
R_B^1	John	Smith	ABVD
R_B^2	Johnathan	Smith	Plavix
R_B^3	Peter	Adams	Lipitor
R_B^4	Paul	Anthony	Actos

Databases C (D_C)

ID	First name	Surname	Plan
R_C^1	John	Smith	PPO
R_C^2	Johnathan	Monette	HMO
R_C^3	Willium	Anthony	HMO

Databases D (D_D)

ID	First name	Last name	Age
R_D^1	John	Smith	46
R_D^2	Johnathan	Monette	32
R_D^3	Willum	Amith	27
R_D^4	Kathy	Anderson	65

Attribute selection for linking of records

Databases A (D_A)

ID	First name	Last name
R_A^1	John	Smith
R_A^2	Johnathan	Smith
R_A^3	Peter	Anthony

Databases B (D_B)

ID	Given name	Surname
R_B^1	John	Smith
R_B^2	Johnathan	Smith
R_B^3	Peter	Adams
R_B^4	Paul	Anthony

Databases C (D_C)

ID	First name	Surname
R_C^1	John	Smith
R_C^2	Johnathan	Monette
R_C^3	Willium	Anthony

Databases D (D_D)

ID	First name	Last name
R_D^1	John	Smith
R_D^2	Johnathan	Monette
R_D^3	Willum	Amith
R_D^4	Kathy	Anderson

Preserve the privacy of individual attribute values (for example using, hash encoding)

Databases A (D_A)

ID	First name	Last name
R_A^1	2d5jght7d	4l5ogh7s
R_A^2	a4f7wt5ry	4l5ogh7s
R_A^3	23jkd57sy	sd7ty245e

Databases B (D_B)

ID	Given name	Surname
R_B^1	2d5jght7d	4l5ogh7s
R_B^2	a4f7wt5ry	4l5ogh7s
R_B^3	23jkd57sy	2dr5t6u8a
R_B^4	st98tc5atl	sd7ty245e

Databases C (D_C)

ID	First name	Surname
R_C^1	2d5jght7d	4l5ogh7s
R_C^2	a4f7wt5ry	2n4dtf78h
R_C^3	k45hdt82s	sd7ty245e

Databases D (D_D)

ID	First name	Last name
R_D^1	2d5jght7d	4l5ogh7s
R_D^2	a4f7wt5ry	2n4dtf78h
R_D^3	k45hdt82s	fg7h3ks8d
R_D^4	as02kdif2	m3d8ex5fa

Link the encoded databases to identify the matching record tuples

Matching Result (Records)

ID	Record Identifiers
R_M^1	R_A^1 R_B^1 R_C^1 R_D^1
R_M^2	R_A^2 R_B^2
R_M^3	R_C^2 R_D^2

Share the sensitive attribute values of the identified matching records

Linked Database

ID	Age	Disease	Drug used	Plan
R_M^1	46	Cancer	ABVD	PPO
R_M^2	–	Diabetes	Plavix	–
R_M^3	32	–	–	HMO

Figure 1.2: Example steps involved in a health surveillance system to link databases from different data sources, such as government departments, hospitals, pharmaceutical companies, and insurance companies, as shown in Figure 1.1.

erally utilised by governments by combining with third-party security agencies or security services [146]. Such an application needs to link transactional data such as store or online purchases, hospital admissions, mobile phone records, airline manifests, hotel registrations, etc. to identify suspicious activities of the identified entities. This requires databases from hotels, airlines, car rentals, banks and so on. to be integrated. Nevertheless, such data integration or matching needs to be performed in real-time because the identification of a suspect is often required in real-time.

However, such a system has to provide two levels of data privacy of the entities in the databases [146]. (1) Since the databases to be matched are very large and diverse, the people who are in the suspect list should not be revealed to any of the database owners or organisations. In most occasions the number of individuals on terrorism watch lists are likely ordinary citizens that show similar characteristics to actual terrorists. Revealing information of such non-terrorist individuals as suspects could violate their privacy because these individuals can be scrutinised when they interact with data sources. (2) Subjects or entities who are not in the suspect list should be protected from these counter-terrorism applications because revealing such sensitive data about innocent citizens to third-party or government agencies for matching and analysis purposes without consent can invade their privacy.

1.1.1.4 Fraud Detection and Prevention

As we are living in the Big Data era, financial and transactional data are generated in many different formats and with huge volumes. Analysing such data requires more sophisticated information systems than traditional methods of data analysis [159]. Traditional methods often require complex and time-consuming investigations to deal with different domains of knowledge like financial, economics, business practices and law. Record linkage is an integral component in modern fraud detection systems that link records across databases from government departments, law enforcement agencies, and financial institutions. Linkage of these databases allows accurate identification of individuals or groups of individuals who are suspected to be fraudsters by verifying the personal credentials they present when they, for example, apply for a credit card or a loan [35, 167, 221].

Fraudsters usually provide fake or modified identification information, such as addresses, dates of birth, or social security numbers, when user credential information is requested by different organisations. Such fake information creates a major challenge in linkage applications, since such data could be viewed as erroneous data that is available in databases due to data entry errors. Furthermore, fraudsters deliberately modify personal details because they do not want to be identified, and they try to make these changes to look like real valid variations [159, 167].

Record linkage provides a way for identifying such falsified identities in databases by linking records across different organisations and matching the credentials provided by each person. This potentially assists in identifying the details provided by an individual as either valid or fake. However, for such a linkage to be employed in fraud detection systems it needs to be capable of matching records in real-time while preserving the privacy of sensitive personal information, as well as being scalable to an increasing number of databases that need to be linked [35, 216].

1.2 Research Problem

When matching multiple databases, potentially each record from one database needs to be compared with all records in the other databases in order to determine if a

Table 1.1: Number of candidate record tuples generated with multiple databases for different sizes (number of records) of databases and blocks.

	Number of databases (d)			
Database size / Block size	3	5	7	10
10,000 / 10	10^6	10^8	10^{10}	10^{13}
10,000 / 100	10^8	10^{12}	10^{16}	10^{22}
10,000 / 1,000	10^{10}	10^{16}	10^{22}	10^{31}
100,000 / 10	10^7	10^9	10^{11}	10^{14}
100,000 / 100	10^9	10^{13}	10^{17}	10^{23}
100,000 / 1,000	10^{11}	10^{17}	10^{23}	10^{32}
1,000,000 / 10	10^8	10^{10}	10^{12}	10^{15}
1,000,000 / 100	10^{10}	10^{14}	10^{18}	10^{24}
1,000,000 / 1,000	10^{12}	10^{18}	10^{24}	10^{33}

pair or a set of records corresponds to the same entity or not. The number of naïve pair-wise record comparisons of multiple databases grows quadratically as the sizes (number of records) of databases to be linked get larger, and exponentially with the number of databases. In such situations, methods or techniques to reduce the comparison space are needed.

In the record linkage process, *blocking* (*indexing*) is generally used to scale the linkage to large databases [36, 165]. Blocking identifies candidate records for comparison and classification by grouping similar records into the same blocks and dissimilar records into different blocks. This enables the record comparison space to only consist of likely true matching record comparisons while removing as many of the true non-matching record comparisons as possible. As a result, blocking of databases decreases the amount of computational efforts required when comparing larger databases. Generally records are grouped according to some criteria (known as a *blocking* or *sorting* key [36]) and only the records in the same block are compared in detail using expensive comparison functions [14, 36].

In recent years, various blocking techniques have been proposed for RL and PPRL (this topic is further discussed in Chapters 2 and 3). However, these blocking techniques are mainly aimed at the linkage of two databases only, which makes them unsuitable in a multidatabase context. In this thesis we focus on the blocking step of the MD-PPRL process (see Chapter 2). Apart from the three key challenges of PPRL presented in Section 1.1, we aim to address the following challenges that are related to blocking in a MD-PPRL context:

- **Challenge 1: Number of databases**. When the number of databases is increasing the naïve record comparison space grows exponentially. For example, with d databases each containing n_r records the total number of record pair comparisons required can be specified as ${n_r}^d$. Nevertheless, as shown by Sadinle and Fienberg [183], separate pairwise matchings of databases do not guarantee the transitivity of the linkage decisions and thus require resolving discrepancies (we will discuss the inapplicability of bipartite matching in a multidatabase linkage context in more detail in Section 4.4 on page 78). This makes multidatabase linkage applications currently not scalable with regard to an increasing number of records and databases.

As shown in Table 1.1, the number of candidate record tuples (tuples of records from different databases that need to be compared, as we will define in Section 4.4.2 on page 81) grows exponentially with the number of databases to be linked for a multidatabase linkage, and even with very small sized blocks (such as 10 records per block per database) the number of candidate record tuples can become prohibitively large. Therefore more sophisticated blocking mechanisms are required for MDRL and MD-PPRL applications.

- **Challenge 2: Database Sizes**. Everyday, organisations generate and store many millions of records in their databases. In the context of these unprecedented volumes of records, existing blocking techniques generate blocks that contain large numbers of records that could potentially require large number of record pairs or sets to be compared. Ultimately, the comparison of these blocks leads to an inefficient linkage.

  However, the generation of a large number of blocks with a small number of records each, or a small number of blocks with a large number of records each, could potentially improve the efficiency of blocking. However, such blocking leads to a poor linkage performance as more record pairs need to be compared. Furthermore, many of these comparisons could potentially generate more non-matches [170]. Table 1.1 illustrates this problem for MD-PPRL which shows the number of candidate record tuples generated for different numbers of databases of various sizes and different block sizes (by assuming all blocks have the same size). Therefore, blocking techniques employed in a multidatabase context should be capable of grouping similar records into the same blocks to minimise the number of potential non-matches while controlling the maximum block size to achieve a highly efficient blocking process.

- **Challenge 3: Data Quality**. In most cases, organisations collect their data through semi-automated processes that cannot filter information of low quality [96, 228]. Often, these collected data contains noise, which includes duplicate records, or records with missing and inconsistent values. In a multidatabase context the effectiveness of the overall linkage is potentially reduced by the variations and erroneous values in records as the diverse nature of data increases with the number of data sources [95]. Therefore, the blocking techniques deployed in a multidatabase linkage should be robust with regard to different types of noise in the databases to be linked.

- **Challenge 4: Collusion**. Privacy is a major concern in any PPRL application which performs linkage on databases from different organisations. In a PPRL context, the parties who are participating in the linkage process can potentially collude with each other to learn about the sensitive data of a party that does not collude. In a multidatabase linkage context the risk of collusion increases with the number of parties that participate in the linkage. This requires the blocking techniques employed in a multidatabase context to be collusion resistant. However, the techniques that are used in blocking to preserve the privacy of sensitive data also need to be computationally efficient and effective to ensure the scalability and quality of the overall linkage.

- **Challenge 5: Redundant record comparisons**. As illustrated in Table 1.1, conducting multidatabase linkage on block collections from different databases still requires a very large number of similarity comparisons because each record in a block needs to be compared with all records in the same block originating from the other databases (that have the same or similar blocking key values). Assuming d databases, each containing n_r records, split evenly into n_B blocks of size $(\frac{n_r}{n_B})$ records results in ${n_B}^d$ potential candidate block tuples (CBTs). We formally define a candidate block tuple in Section 4.2 on page 65. The pair-wise comparison of these CBTs will result in $n_B \cdot (\frac{n_r}{n_B})^d$ record pair comparisons. However, a large portion of these candidate block comparisons between different databases consists of redundant record pair comparisons that occur due to *repetitive* and *superfluous* comparisons of record pairs, where the latter are comparisons with record pairs that have previously been classified as non-matches [164]. Such redundant record comparisons involve unnecessary computation costs in classification and comparison techniques in the linkage process. Removal of these redundant comparisons in the blocking stage not only improves the effectiveness (reduces non-matches), but also increases the efficiency (reduces the number of record pair comparisons) of the overall linkage.

  Therefore, novel blocking techniques are required that process each candidate block tuple by identifying and purging unnecessary comparisons in multidatabase linkage in order to increase the overall efficiency. Furthermore, these techniques should not affect the overall effectiveness of the linkage by removing truly matching record sets.

- **Challenge 6: Subgroup structures**. The primary goal of MDRL and MD-PPRL is to identify similar records that exist across all the databases. However, in real-world applications, being able to identify matching records that occur in two or more databases is important for decision making activities [22]. The sets of matching records across subgroups of databases can be valuable for conducting analytical studies and decision making about sub-populations or groups that exist within a larger population [41, 131].

  In order to identify subgroups of records that are matching across multiple databases, the linkage between records from different subgroups of these databases is required. However, as the number of databases to be linked increases the number of subgroup combinations of different sizes increases exponentially. For example, with d databases the potential number of subgroups is equal to $\sum_{n=1}^{d} \binom{d}{n} = 2^d - 1$. Table 1.2 shows the potential number of subgroups of sizes 2 to d that can be generated for different number of databases. Furthermore,

Table 1.2: Number of subgroups of sizes 2 to d possible across different number of databases.

Number of databases (d)	Number of possible subgroup combinations
3	4
5	26
7	120
10	1013

the use of existing blocking techniques multiple times for linking subgroups of different sizes of databases is not computationally feasible. Therefore, blocking techniques for multidatabase linkage should be capable of identifying similar records in subgroups of databases efficiently.

We focus on developing novel blocking approaches for MD-PPRL that go beyond existing blocking techniques by addressing the challenges specified above. In the following section we describe the aims of our research in more detail, while in Sections 1.4 and 1.5 we describe the contributions arising from this thesis and the research methodology we followed to address these challenges, respectively.

1.3 Aims and objectives of this research

As we have described in Section 1.2, the comparison of records of multiple databases requires significant computational time and resources. Importantly, the techniques used in MD-PPRL should be scalable to an increase in the size and number of databases to be linked. Furthermore, the techniques used in a MD-PPRL application should provide enough privacy for sensitive data in those databases to be linked as the collusion risk increases with the number of parties involved in a linkage.

During the past few decades many approaches have been proposed to perform record linkage between two databases [36, 71, 184]. However, as explained in Section 1.2, the linkage between multiple databases is still an open research problem that requires further investigation. As many real applications require data to be integrated from multiple sources, this demands the development of scalable technique in the multidatabase linkage context. Also, these techniques should be able to achieve high linkage quality without compromising the privacy of the sensitive data in those databases.

The main aim of this thesis is to design and develop blocking techniques that can be applied in a MD-PPRL context. Our aim is to develop techniques that can scale with the size and number of databases to be linked, as described in Section 1.1. We aim to use different privacy techniques (as detailed in Chapter 2) to develop efficient blocking techniques that help to preserve the privacy of the databases to be linked. Based on our aims we define several objectives to accomplish in this study:

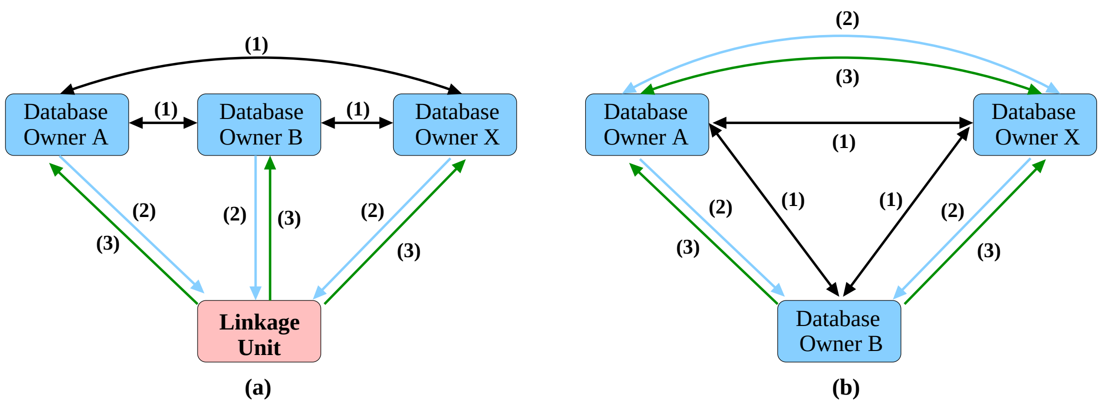


Figure 1.3: Multidatabase linkage models, (a) with a linkage unit and (b) without a linkage unit. In these models the communication among the participating parties can be given as: (1) exchange of parameter values, (2) communicate the (somewhat) encoded records of databases, and (3) exchange or send the linkage result.

1. **Develop blocking techniques that can be applied in different MD-PPRL linkage models**. In a multidatabase context the linkage of different databases can be performed with or without a *linkage unit* (LU) (more details of this topic are given in Chapters 2 and 3) as illustrated in Figure 1.3 [22, 176, 219]. A LU is a special party or organisation that participates in a protocol to conduct the linkage of the data sent to it by the owners of the databases. We define the LU in more detail in Section 2.4. In a privacy-preserving context the common communication steps that can occur between the participating parties within these linkage models can be given as: (1) exchange of parameters, pre-processing standards, secret keys, etc. between the database owners, (2) exchange of the encoded (masked) values in the databases, and (3) exchange of the linkage results. In this thesis we develop blocking techniques under both these linkage models shown in Figure 1.3.

   As shown by Mitchell et al. [149], most of the practical linkage applications, such as Microdata Linkage at Statistics Canada[5], E-health informatics research in UK[6], and health data linkage in Australia[7], use LUs (data linkage centres) to facilitate and conduct the linkage of different databases compared to database owners performing the linkage by themselves. As we describe in Section 2.3, the linkage of databases without a LU can become more computationally and communicationally expensive compared to the linkage applications with a LU because of the use of more sophisticated encoding or encryption mechanisms [66]. However, linking records at a central location can potentially

[5]https://www.statcan.gc.ca/eng/record/gen
[6]http://www.cancerresearchuk.org/funding-for-researchers/how-we-deliver-research/our-research-partnerships/the-farr-institute-of-health-informatics-research
[7]http://www.cherel.org.au/how-record-linkage-works

increase the risk of privacy attacks because the LU can use the frequency information of these encoded values to deduct a sensitive plain-text value of a database. We explain privacy attacks in Section 2.4.5.

Under these linkage models the privacy of the data needs to be considered for different adversary (security) models as described in Chapter 2. First we investigate and identify which PPRL techniques are more suitable for each linkage model and each adversary model. By doing so, a good understanding can be gained about MD-PPRL techniques and their limitations, efficiency, and effectiveness under different adversary models. This will help us to develop different blocking techniques under these linkage models.

2. **Develop blocking techniques that are more collusion resistant**. As we described in Section 1.2, the potential risk of collusion between participating parties (including database owners, linkage units, or any external organisations participating in the linkage) increases with the number of parties. This increases the privacy risk in blocking since a group of participating parties can collude with each other in the blocking step to identify sensitive record values in a database of another database owner that does not collude. Therefore, we investigate the applicability of different techniques to improve the privacy in our multidatabase blocking approaches.

3. **Develop blocking techniques that support subgroup blocking**. In a multidatabase environment, subsets of matching records could also be available in two or more databases to be linked. These subsets of matching records provide important information about the specific entity groups that exist within the whole population. However, to identify subsets of matching records, we need to identify candidate blocks that need to be compared for different subgroups across all the databases to be linked. Therefore, we focus on developing approaches that support subgroup blocking in a multidatabase linkage scenario.

4. **Efficiently remove redundant record set comparisons**. As shown in Table 1.1, the record comparison space in a multidatabase linkage results in an excessively high number of candidate record tuples that need to be compared even when a blocking technique is applied. However, the record comparison space generated by a blocking technique could encompass redundant record comparisons that are repetitive and/or superfluous. Our objective is to develop filtering techniques that can be incorporated with the blocking step for multidatabase linkage to remove these redundant record comparisons. This helps to maximise the overall efficiency while retaining the original or high levels of effectiveness in the overall linkage.

1.4 Contributions

This thesis is concerned with PPRL on multiple databases. We focus mainly on the blocking step of the PPRL process (as described in Section 2.3 on page 22) and propose solutions to address the main challenges involved in blocking of multiple databases in a privacy-preserving context. Our contributions can be grouped into six main sections:

1. **A blocking framework for MD-PPRL**: A major shortcoming in the MD-PPRL context is that a blocking framework that could address the challenges related to multidatabase blocking has so far not been developed. This requires database owners or PPRL practitioners to use PPRL techniques that are not applicable in real-world applications. To address this issue we propose a blocking framework for MD-PPRL that goes beyond existing solutions in three different ways:

   - our framework is designed to deal with the challenges of MD-PPRL (as detailed in Section 1.3),
   - it divides the blocking process into different steps, which allows users to maximise the overall efficiency and effectiveness of the linkage of multiple large databases, and
   - it provides PPRL practitioners and users with different techniques, approaches, and protocols to create highly scalable MD-PPRL solutions that can be easily tailored to the particular settings and requirements of a given application.

   The blocking techniques that are developed in our proposed framework are grouped into different layers. Each layer is responsible for a specific aspect of blocking in the MD-PPRL process and receives as input the output of the previous layer. The goal of this framework is to produce an output from each layer that improves the effectiveness and/or the efficiency of the input provided by the previous layer. This proposed blocking framework is presented in Chapter 4.

2. **Scalable blocking techniques for MD-PPRL**: As discussed in Section 1.3, we propose several blocking approaches for MD-PPRL. We use a multidatabase linkage model without a linkage unit, as shown in Figure 1.3 (b), in these proposed approaches as it provides more privacy in generating blocks. We propose three blocking techniques that use an efficient tree data structure and two clustering techniques for generating blocks of multiple databases that need to be linked. The proposed blocking techniques are described in Chapters 5 and 6.

3. **A distributed blocking approach for MD-PPRL**: As will be detailed in Chapter 3, most blocking techniques proposed in the literature do not provide the database owners (DOs) with flexibility and control over the blocking process. To address this issue, we propose a blocking technique that allows database owners to perform their blocking process independently and then identify the blocks that need to be compared. Our blocking approach follows the multidatabase linkage model with a linkage unit, as shown in Figure 1.3 (a), where we use the linkage unit to identify the block tuples that need to be compared. This contribution is described in Chapter 7.

4. **A subgroup blocking approach for MD-PPRL**: As detailed in Section 1.2, a major challenge in MD-PPRL is to identify records across different databases that need to be compared across subgroups of these databases. To address this problem we propose a subgroup blocking technique that groups records into blocks and identifies candidate blocks from different databases that need to be compared for subgroup combinations of different sizes across those databases. The proposed technique consists of a graph data structure based candidate block generation method for identifying blocks that need to be compared for different subgroup combinations. The proposed approach is presented in Chapter 8.

5. **A meta-blocking approach to reduce redundant record comparisons**: In any multidatabase linkage application, a major portion of the record comparison space consists of repeated and superfluous record comparisons, as we detailed in Section 1.3. We propose a non-parametric meta-blocking approach that can be employed between the blocking and the comparison and classification steps of the MD-PPRL pipeline. The proposed approach efficiently schedules block pairs for comparison ensuring repeated and superfluous record comparisons are eliminated without reducing the effectiveness of the overall linkage. Our approach is designed in such a manner that it can be incorporated into any RL or PPRL applications. The proposed contribution is presented in Chapter 9.

6. **A comprehensive evaluation of the proposed techniques**: We conduct a comprehensive experimental evaluation of our proposed techniques in terms of scalability, blocking quality and privacy using multiple large real and synthetic datasets. We provide a comparative evaluation of our proposed approaches and several state-of-the-art blocking techniques [36, 56, 119] for different MD-PPRL scenarios. The results are presented in Chapter 10.

In addition, all proposed approaches presented in this thesis are conceptually analysed with regard to complexity, blocking quality, and privacy, which are detailed in each corresponding chapter.

1.5 Research methodology

In this thesis the used methodology for designing and developing algorithms for MD-PPRL consists of several steps, as shown in Figure 1.4:

1. Preliminary study: In this step a basic understanding about record linkage was gathered by broadly studying the literature which helped to recognise different research problems in the record linkage context.

2. Formulating research problem: In this step we defined the research problem under the MD-PPRL context.

3. Literature review: This step consisted of an extensive and ongoing study of the current literature available on record linkage, which helped to better understand the research problem. The study was conducted in two streams to

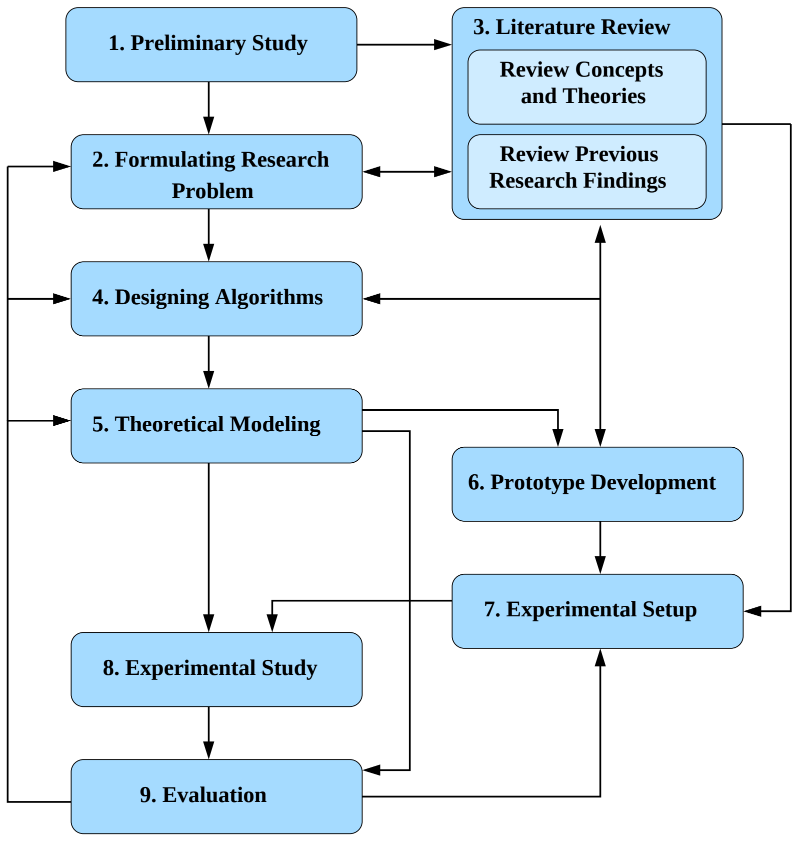


Figure 1.4: The used research methodology.

understand about, (1) concepts and theories and (2) previous research findings in the context of record linkage. This knowledge was used to refine the research problem, design solutions, implement prototypes, and experimental design.

4. Designing algorithms: In this step new approaches were proposed to address the identified research problem.

5. Theoretical modeling: In this step the proposed approaches were theoretically analysed in terms of scalability, blocking quality, and privacy.

6. Prototype development: Prototypes of the proposed approaches were developed to be used as proof-of-concept implementations. We developed all the prototypes using the Python programming language [142] as separate modules. More information on prototype development is given in Section 4.4 on page 84.

7. Experimental setup: In this step the setup of experiments to be run on the developed prototypes was constructed by selecting suitable datasets, evaluation metrics, parameter settings, and selecting other state-of-the-art techniques for comparisons.

8. Experimental study: The set of experiments on the developed prototypes were conducted to gather results for the experimental evaluation.

9. Evaluation: The experimental results were evaluated with regard to scalability, blocking quality, and privacy to validate the theoretical analysis.

1.6 Thesis Outline

This thesis is structured as follows. In the following chapter we provide the background about overall privacy-preserving record linkage (PPRL), and its concepts. Then, in Chapter 3, we present a review of the literature of existing PPRL approaches. We propose our blocking framework in Chapter 4 and the techniques used in our framework are detailed in the following chapters. Chapters 5, 6, and 7 provide details on different blocking techniques for MD-PPRL, while Chapter 8 details our subgroup blocking technique. A novel meta-blocking protocol for further reduction of the number of record comparisons is then presented in Chapter 9. In Chapter 10 we provide a comparative evaluation between the proposed approaches and several existing blocking techniques. Finally, Chapter 11 concludes the study, summarises the achieved results, and discusses directions for future work of this research area.

Chapter 2

Background

In this chapter, we present background knowledge for the following chapters of this thesis. In Section 2.1 we describe the traditional record linkage (RL) process. We provide a detailed description about privacy-preserving record linkage (PPRL) in Section 2.2, and describe each step in the PPRL process in more detail in Section 2.3. In Section 2.4, we provide a detailed description on different privacy aspects related to PPRL, and we describe evaluation measures used in PPRL in Section 2.5.

2.1 Record Linkage

Linking records of different databases has a long history. The term *Record Linkage* was first introduced by Halbert L. Dunn in 1946 [55]. The article he published in the *American Journal of Public Health*[1] describes an idea of creating a book named *book of life* for every individual in the world. Each book of an individual person contains pages covering the principle events of a life, such as the individuals' contacts with the health and social security systems from birth to death. In his article, Dunn highlighted the importance of having such records about individuals for government statistics and services, and for the identification of individuals. However, collection of such data in real-world could become difficult due to data quality issues.

Over the past few decades an increasing interest has been seen into research in the area of record linkage or data matching in domains such as computer science, statistics, and the health and social sciences. The probabilistic foundation of modern record linkage has been proposed by Newcombe et al. [156]. In their work phonetic encoding [35, 66] is applied to attribute values to overcome variations in the data. A calculation based on the distribution of records was used to group records into matches and non-matches. This idea was then formalised by Ivan Fellegi and Alan Sunter who proved that a probabilistic decision rule can be optimal when the comparison of quasi identifier (QID) attributes [94], such as first name, last name, address details, age, etc., are conditionally independent [71]. Even today their work remains the mathematical foundation for many record linkage applications [28, 61, 88, 127, 147, 178, 183, 184, 226, 227].

[1]http://ajph.aphapublications.org/

As organisations, including banks, government departments, and businesses, collect large quantities of data in their databases, linking records from different databases to improve the quality of data analysis and data mining has become increasingly important [22, 66, 228]. Example application areas are national census, health-care, crime and fraud detection, national security and business applications as described in Chapter 1. In national census, record linkage is used as an important tool for census statistics which can help to reduce costs and efforts for conducting large scale census collections [77, 82, 181]. In the health-care sector, linking data from multiple databases can generate important information which is required in drug management, disease prevention, and disease outbreak detection [22, 24, 123, 157]. To detect unusual patterns within and across a variety of databases, record linkage is used in the national security domain [106, 221] which is challenging due to the heterogeneous nature of data in those databases, including database schemas, data formats, and data quality (errors and inconsistencies).

In general, record linkage is a classification problem where record tuples (pairs or sets) from different databases are grouped into the two classes of *matches* and *non-matches*, or into three classes which also include *potential matches* as the third class [36, 71]. The class of matches contains the record tuples that refer to the same real-world entity (a person, an organisation, or an object), while the class of non-matches refers to those record tuples that do not refer to the same entity. The class of potential matches refers to record tuples that cannot be automatically classified as either matches or non-matches. A clerical review process is required to manually classify the records in the class of potential matches as matches or non-matches. The process of performing this classification in any record linkage application is challenging because of the two primary requirements of scalability and linkage quality [35, 66].

2.2 Privacy-preserving Record Linkage

On occasions where unique identifiers for entities are available across all the databases to be linked, a simple database join would be trivial for the purpose of identifying the matching pairs of records. However, due to the lack of unique entity identifiers, the linkage of databases generally needs to be based on the personal details, such as names and addresses, of the individuals whose records are stored in these databases. These attributes generally allow to accurately link records. However, if the databases to be linked belong to different organisations, such linkage will reveal personal information of a database to other organisations that participate in the linking process, which raises privacy and confidentiality issues [23, 29, 217].

During the past decades the challenge of protecting the privacy of personal information, such as names and addresses of people, has gained significant attention in many application domains. Often privacy and confidentiality regulations prevent the owners of databases from sharing any sensitive details of their records with any other organisation. The Beyond 2011 Programme [75] established by the Office for National

Statistics (ONS) in the UK for the production of population and socio-demographic statistics has indicated the importance of anonymisation of data to ensure privacy. As we detail in Section 2.4, data anonymisation is the process of either encrypting or removing personally identifiable information from databases, so that the entities or individuals who are described by the data remain anonymous. Data can be considered anonymised when it does not allow re-identification of the individuals to whom it relates, and it is not possible that any individual can be identified from the anonymised data by any further processing or by processing it together with other (public) information which is available or likely to become available in future [67].

Recently, the European Council has made pseudonymisation of record linkage identifiers factually mandatory. In contrast to data anonymisation, data pseudonymisation substitutes or replaces any identifying characteristics of data of an individual or an entity with a pseudonym in such a way that additional information is required to re-identify the individual so that an entity cannot be identified directly by using such data [203]. Due to increasing privacy concerns, in 2016 the European Council Parliament and Commission agreed on a new *General Data Protection Regulation*[2] that demands pseudonymisation techniques on personal data before any personal information can be used on any projects [67]. This regulation requires all 28 member states of the European Union to implement rules in their national jurisdictions by May 2018.

The Data-Matching Program Act[3] and Australian Commonwealth Privacy Act (1988)[4] in Australia, the Health Insurance Portability and Accountability Act (HIPAA)[5], Children's Online Privacy Protection Act (COPPA)[6], Gramm-Leach-Bliley Act (GLB)[7], and Privacy Act of 1974[8] in the USA, the Data Protection Act[9] in the UK, and the Privacy Act 1985[10] in Canada are some examples that describe the legal restrictions of disclosing private or sensitive data across organisations.

In the last two decades the increase of data privacy concerns in organisations has led to a significant attention in the research community to further investigate how to link records across organisations without compromising the privacy of the entities represented by these records [189, 194, 216]. The research paradigm of finding records in multiple databases that relate to the same entity or having approximately the same values for a set of quasi identifiers, such as names and addresses, without revealing any private or sensitive information is known as *privacy-preserving record linkage* (PPRL) [36, 88, 189, 218], *private record linkage* [3, 13, 98, 224, 231], or *blindfolded record linkage* [42].

[2]http://www.eugdpr.org/
[3]http://www.oaic.gov.au/privacy/privacy-act/government-data-matching
[4]https://www.oaic.gov.au/privacy-law/privacy-act/
[5]http://www.hhs.gov/ocr/privacy/
[6]http://www.inc.com/encyclopedia/childrens-online-privacy-protection-act-coppa.html
[7]http://www.ftc.gov/tips-advice/business-center/privacy-and-security/gramm-leach-bliley-act
[8]https://www.justice.gov/opcl/privacy-act-1974
[9]https://www.gov.uk/data-protection
[10]http://laws-lois.justice.gc.ca/eng/acts/p-21/

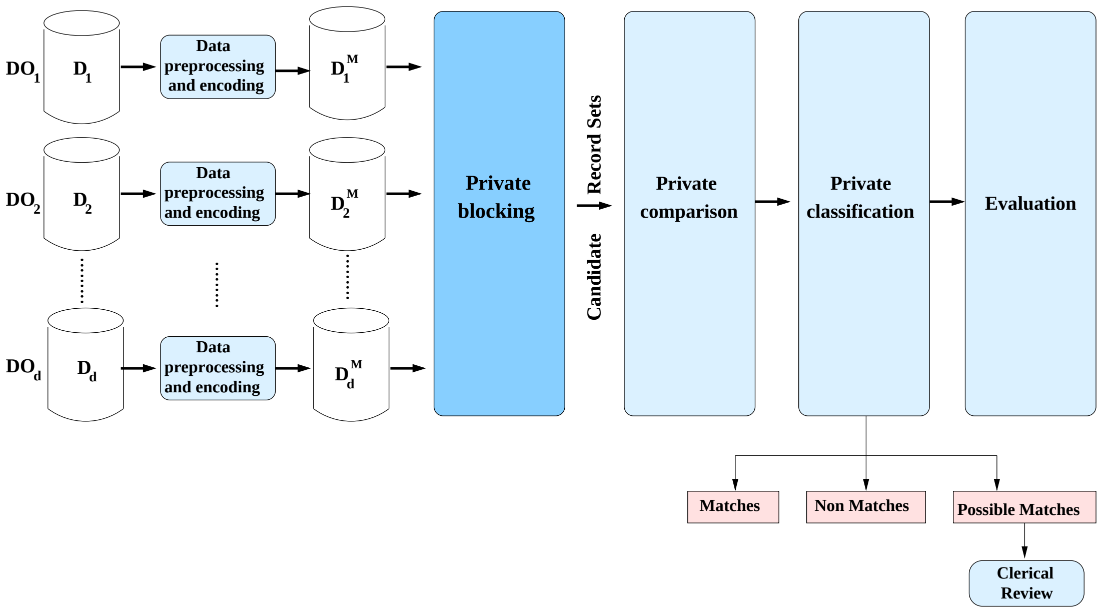


Figure 2.1: Outline of the privacy-preserving record linkage pipeline as detailed in Section 2.3 for linking d databases owned by different databases owners (DOs). This thesis mainly focuses on the emphasised blocking step of the PPRL process which is showed in dark blue.

In PPRL, the linkage is based on different privacy techniques, such as encoding, to ensure the privacy of data. We will describe different privacy techniques used in PPRL in Section 2.4. As we discussed in Section 1.1, the database owners need to reveal some information (records that have been classified as matches, or a selected set of attribute values from the matched records) at the end of the linkage process. However, each step in the record linkage process needs to preserve the privacy of the sensitive details of the data to be linked [23, 189, 216, 218]. The following section provides more details about the steps involved in the PPRL process.

2.3 The PPRL Process

Figure 2.1 illustrates the main steps involved in the PPRL process for d databases. The PPRL process has the same set of steps as in a traditional record linkage process, however, data privacy needs to be considered in each step [212, 217, 219].

Since most real-world databases contain incomplete and inconsistent data [169, 174], data pre-processing often has to be applied first in the PPRL process. To improve linkage quality, the databases to be linked must use the same process of data cleaning and standardisation [43]. As illustrated in Figure 2.1, each database owner (DO) can conduct the data pre-processing step independently while agreeing upon the techniques to be used for pre-processing and the set of attributes to be used for linkage.

Once the databases are pre-processed, the values in the attributes used for the linkage of the records need to be encoded (masked or encrypted) to ensure no sensitive information that can be used to infer individual records and their attribute values is revealed to any party that participates in the linkage process, or to any external adversary [212, 218]. Each DO needs to agree on the encoding technique and relevant parameter settings to be used to ensure the same encoding process is applied across all the databases to be linked. As shown in Figure 2.1, each database $\mathbf{D}$ is encoded into a masked database $\mathbf{D}^M$ using an appropriate encoding technique. We will discuss different encoding techniques used in PPRL in Section 2.4.

As we have described in Section 1.2, the naive pair-wise comparison of multiple databases is of exponential and quadratic complexity in terms of the number and size of databases, respectively, which makes PPRL applications not scalable to linking multiple databases [170, 210]. Blocking is used in the linkage process with the aim to improve scalability [36, 165]. Blocking reduces the number of potential record comparisons by removing as many record tuples as possible that correspond to non-matches, such that expensive similarity calculations are only required on a smaller number of record pair comparisons. In PPRL, blocking needs to be applied and conducted on encoded attribute values to ensure no participating party can infer individual records and/or attribute values [212, 218].

As shown in Figure 2.1, after the blocking step is finished the records of blocks from different databases, called *candidate record tuples* (to be formally defined in Chapter 4), need to be compared in the comparison step. The similarities between the corresponding attribute values of the candidate record tuples are computed by using approximate comparison functions, such as edit distance, Jaccard similarity, or Jaro-Winkler similarity [33, 47, 226].

However in PPRL, due to the encoding process similar attribute values could generate completely different encoded values due to errors and inconsistencies in those values. Comparison of such encoding values often requires approximate comparison functions rather than exact comparison to compute their similarities appropriately. By assuming a set of common attributes has been used in the linkage across all databases, this step outputs a similarity vector for each candidate record tuple [210].

As discussed in Section 2.1, the aim of linkage is to classify the generated candidate record tuples into the classes of matches, non-matches, or possible matches. In the classification step the similarity vectors generated for candidate record tuples are classified using an appropriate decision model [85], as we have defined in Definition 1.1. In RL, different classification techniques have been developed. These include threshold based [35], machine learning [6, 16, 34, 66, 222], rule based [155], and probabilistic [71, 127, 183, 227] classification techniques. However, mostly threshold based classification techniques have been adapted for PPRL so far [176, 195, 210, 214, 224].

The classification technique that is used in the PPRL process should output only the record tuples that have been classified as matches. This ensures the privacy of the records that do not match. Also, the similarity values of the compared record tuples and the similarity distribution across all compared record tuples should not be revealed to any party that participates in the linkage because of possible privacy attacks [23, 216]. We will provide details about different privacy attacks in Section 2.4. As illustrated in Figure 2.1, the final step of the PPRL process, evaluation, measures the performance of a linkage application in terms of efficiency and effectiveness. Efficiency provides a quantitative measure on the scalability of the techniques used in a linkage process [35]. Effectiveness measures the accuracy of the classification performed by the linkage process.

For evaluating the scalability and linkage quality different measures can be used (we present linkage quality measures for MD-PPRL in Section 4.4.2). Evaluation of linkage quality is more challenging in a privacy-preserving context because accessing the actual record values would likely reveal sensitive private information. Evaluating the amount of privacy protection provided by a PPRL technique is the least developed aspect in PPRL research [212]. More details of these evaluation techniques and metrics are given in Section 2.5.

2.4 Privacy Aspects in PPRL

This section provides a detailed description of the privacy aspects that need to be considered when performing record linkage between different organisations.

2.4.1 Role of Parties

In a PPRL context, depending on the linkage model used, different parties can participate in the linkage process. The roles of these parties can be categorised as:

- **Database owner**
  The database owners (DOs) are the providers of the databases to be linked [35]. According to the linkage model employed, the database owners may or may not participate in the record matching or comparison step (where databases are sent to another party, such as a linkage unit, to perform the linkage) . Financial institutions, medical service providers, or government departments are some examples of database owners. Database owners are also known as data custodians [212].

- **Linkage unit**
  A linkage unit (LU) is a special party that participates in the linkage process such that it may or may not be external to the DOs [22, 23, 176, 196, 209]. In general, a LU does not have any data itself but conducts the linkage of the data sent to it by the DOs. In PPRL various protocols have been developed that use a LU to conduct the linkage. These protocols use different privacy techniques, as we will describe in Section 2.4.4, to protect the privacy of individuals and to maintain the security of data that are communicated between the participating parties and the LU [23, 219].

- **Global facilitator**
  A global facilitator (GF) is someone who helps a group of DOs to understand their common objectives and assists them to plan on how to achieve these objectives in the linkage process [143]. The objectives can include the communication and exchange of parameters and attribute selection among the database owners. A GF is not involved in the actual linkage process.

- **Global Authority**
  A global authority (GA) is another special party which plays the role of a central public semi-trusted regulatory agency in a linkage model [114, 119]. A GA can create a global summary (or view) about the data at some stage of the protocol which can be used by the DOs to make decisions about the next steps in the protocol. For example, in [114] a GA is used to generate global synopses to compute the intersection of different sets of different parties (we explain this approach in more detail in Chapter 3). Some GAs, called as *key authorities*, can be used to generate the necessary secret keys (public and private) for cryptographic functions (encryption and decryption) used in the linkage process [120].

- **Data Consumers**
  Data consumers are the data users in the linkage process [17]. After the linkage process is completed, the matching or linkage results (such as record identifiers or values from a few selected non-identifiable attributes) are sent to data consumers. Database owners, data analysts, and external researchers can be considered as data consumers as shown in Figure 1.1.

2.4.2 Number of Parties

By assuming each database that is to be linked is owned by a different owner, a linkage protocol that is used in a privacy-preserving context can be categorised based on the number of databases that are to be linked.

- **Two database protocols with a LU**
  In this scenario two DOs are performing PPRL on their databases through a LU [23, 28, 111, 116, 176, 209, 224]. In general the two DOs start the process by sharing any required information such as parameters, pre-processing methods, encoding methods, secret keys etc. After the records are encoded they will be sent to the LU to perform the matching by calculating the similarities between the encoded records. The matching results are sent back to the DOs to either exchange the details of the matched records or to send only a selected set of attribute values of the matched records to a data consumer.

  However, the LU can collude with a DO to infer the attribute values for the other DO, though these protocols are more efficient due to centralised processing of records. Also, sending all encoded records to the LU can potentially increase the risk of privacy attacks because the LU can use the frequency information of these encoded values to deduct a sensitive plain-text value of a database. We explain privacy attacks in Section 2.4.5.

- **Two database protocols without a LU**
  Two database protocols are proposed as an alternative to overcome the general drawback of requiring a LU in the linkage process. In two database protocols the DOs directly communicate with each other to perform the matching of their records [99, 137, 208, 215, 231]. This will make the protocol more secure compared to two database protocols with a LU because no collusion (as described in Section 2.4.5) can occur between the DOs. However, two database protocols can become more complex and expensive in terms of computation and communication due to the use of more sophisticated encoding or encryption mechanisms [66].

- **Multidatabase protocols with a LU**
  As shown in Figure 1.3 (a) on page 12, in these protocols more than two DOs are participating in a linkage process by using a LU for performing the linkage [108, 127, 160]. Similar to three-party protocols, the DOs have to agree on the set of parameters, set of attributes, and pre-processing methods which will be used in the linkage process. However, the linkage process needs to be conducted using more scalable and efficient techniques due to the exponential growth of the comparison space with the number of databases to be linked and their sizes.

- **Multidatabase protocols without a LU**
  Similar to two database protocols without a LU, the security of multidatabase scenarios can be increased by removing the LU in the linkage process [135, 152, 210]. With an increased number of databases involved in the linkage more sophisticated computation and communication techniques are required to guarantee the efficiency and privacy of the attribute values in the databases. An example of a multidatabase protocol without a LU is shown in Figure 1.3 (b).

In general, multidatabase protocols are more susceptible to collusion risks as DOs can collude with each other or with the linkage unit to learn about the sensitive data of one or more DOs that are not colluding [212]. Therefore, more secure as well as more efficient linkage techniques are required for MD-PPRL scenarios. Hence, this thesis concentrates on developing multidatabase blocking approaches for the last two types of protocols. In Chapters 5 and 6 we propose several blocking techniques for MD-PPRL protocols without a linkage unit and in Chapters 7, 8, and 9 we propose different blocking techniques that can be employed in MD-PPRL with a linkage unit.

2.4.3 Adversary Models

In the protocols discussed in Section 2.4.2, the parties (DOs and the LU) are assumed to behave in one of several different adversary models.

- **Honest-but-curious (HBC) adversary**
  Most of the PPRL protocols proposed in the literature follow the *honest-but-curious* (HBC) adversary model [3, 170, 208, 214]. This model is also known as the *semi-honest* and *passive* adversary model [88, 140]. In this model, all parties that participate in a linkage follow the steps of the protocol, but they are curious in learning about the data of another party [83, 215]. While following the steps, a DO or the LU can store information and the results it received from other DOs. This information can be used to infer sensitive attribute values of a database by using a frequency analysis or cryptanalysis technique [39, 132]. For example, a DO can perform a frequency analysis on the received data to get a frequency distribution of sensitive attribute values.

  A linkage protocol can be considered as secure under the HBC adversary model only if all parties that participate in the protocol cannot infer any sensitive information other than the records classified as matches from the protocol. However, under the HBC adversary model collusion between the parties is possible where several DOs can use the information they received during the protocol together to learn about the sensitive data of another DO [140]. We will explain collusion in more detail in Section 2.4.5.

- **Malicious adversary**
  The malicious adversary model assumes that the parties involved in the protocol may not follow the steps of the protocol, rather they can behave arbitrarily [88, 140]. In the protocol steps a malicious DO can send malicious data to learn about any sensitive information of another DO. PPRL protocols developed under the malicious adversary model mainly use *secure multi-party computation* (SMC) techniques [233] which use encryption and encoding techniques to ensure no malicious party can learn any sensitive information of any other party [137, 152]. This makes PPRL protocols under the malicious adversary model to have higher communication and computation complexities compared to the linkage protocols which follow the HBC model.

- **Covert adversary**
  Typically, in most PPRL protocols the participating parties are assumed to be following either the HBC or malicious adversary models. However, in many real-world settings, the assumption regarding the HBC behaviour does not suffice, while security in the presence of malicious adversaries is excessive and expensive to achieve. The covert adversary model lies between the semi-honest and malicious models, where it matches with many real-world settings by assuming parties may deviate arbitrarily from the protocol specification in an attempt to cheat until they are being caught [10, 140]. This suggests that a DO

who is cheating in a protocol can be caught by an honest DO with a given probability. This probability is known as the *deterrence factor* [140]. For example let us assume the deterrence factor is equal to ϵ. Then, any attempt to cheat by an adversary is detected by the honest parties with probability of at least ϵ. The higher the value of ϵ, the greater the probability that the adversary is caught and thus the greater the deterrent to cheat. The protocols developed for this adversary model generally use computationally more expensive oblivious transfers and homomorphic encryption schemes to perform the communication between parties [10, 11].

- **Accountability computing**
  Accountability computing (also called *accountable computing*) is a framework which is practical for many real-world applications without the complexity and computational cost required for a linkage protocol that uses SMC techniques under the malicious adversary model [103, 139]. The idea behind accountability computing is that a DO that correctly follows the protocol can be proven to have done so and consequently it can be proven if some other DO has disclosed data or deviated from the protocol. This enables practical utility over HBC protocols while providing detection capability over the malicious model.

2.4.4 Privacy Techniques

In a privacy-preserving context, different privacy techniques can be employed in the record linkage process to preserve the privacy of data and to prevent re-identification of individuals (entities) represented by such data. These techniques can be categorised into two main groups as follows.

1. **Perturbation methods**
   Perturbation or data sanitisation methods modify sensitive private data to prevent re-identification of individuals by inferring records or attribute values. Some of the techniques that can be given under this category are:

   - **Embedding approaches**
     In embedding approaches attribute values are mapped to objects in a multi-dimensional Euclidean [19] or Hamming [115] space, whereby the similarities between the attribute values are preserved. Embedding approaches have been used in several PPRL approaches [19, 186, 231], however, one drawback with these approaches is that it is often difficult to determine appropriate dimensions of the multi-dimensional space.

   - **Bloom filters**
     Bloom filter is a space efficient probabilistic data structure proposed by Bloom [18] in 1970 for checking element membership in a given set [26]. A Bloom filter is a bit vector where elements in a set are hashed or mapped into using a set of hash functions. When an item is queried from a Bloom filter, *false positives* [35] can occur, however, *false negatives* [35] are always

not possible. As more elements are added to a Bloom filter, the probability of false positives increases [150]. We describe Bloom filters in more detail in Section 4.3.

In 2009, Schnell et al. [190] proposed a approximate matching technique for textual (string) values using Bloom filters for PPRL, while Vatsalan and Christen [211] recently proposed an approximation matching technique for numerical attributes, as we describe in Chapter 3. Recently, Bloom filters have been widely used in the PPRL context as they provide considerable privacy guarantee [28, 53, 59, 166, 190, 192, 194, 208, 212, 224]. In this thesis, we use Bloom filters as a building block for our blocking techniques proposed in Chapters 5 and 6.

- **Count-min sketches**
  Count-min sketches are probabilistic data structures that can serve as frequency tables of values in a stream of data [49]. They use a set of hash functions to map values along with their frequencies using sub-linear space. Count-min sketches are similar to counting Bloom filters (that are integer vectors that contain a count value in each position than a 1 or 0) [69], however, typically they have a sub-linear number of cells in their table of values. Recently, count-min sketches have been used in PPRL to identify the frequency of occurrences of matching record pairs [114].

- **k-anonymisation**
  k-anonymisation [200] is a data generalisation mechanism, which has been used widely in the PPRL context [107, 111], to overcome the problem of re-identification of individual records by generalising the sensitive data in such a way that re-identification from the perturbed data is not feasible. In k-anonymisation a database is said to have the k-anonymity property if an individual that is represented by a record in the database cannot be distinguished from at least $k - 1$ individuals whose information also appears in a database. The idea behind k-anonymisation in PPRL is that a given record in a database cannot be re-identified from a group of at least k records where the other records also share the same values (every combination of (masked) quasi-identifier values) in a certain set of attributes as the given record [200].

- **Use of public reference values / random values**
  Several PPRL approaches have used reference values from public databases or values made-up randomly to calculate distances between attribute values and reference values and share these distances to identify the matching pairs of records [186, 214, 215, 231].

- **Differential privacy**
  Differential privacy has emerged as an alternative to randomisation noise addition technique for PPRL that provides strong privacy guarantees [62]. Initially, differential privacy was designed to support interactive queries and aggregate results presentation by adding noise to each statistical query

result (such as Count or Sum) with the magnitude of noise depending on a privacy parameter ϵ and sensitivity of a given query set [63]. The main idea in differential privacy is that an adversary that attempts to attack the privacy of some individual entity will not be able to distinguish from the interaction with a database whether a record representing that entity is available in this database or not [19, 30, 99, 134]. In a context of PPRL each database owner could use a randomised sanitisation scheme on their database in order to achieve differential privacy. To improve the privacy of individual records a certain amount of random noise is added to each query result. Then, only the perturbed result of a set of statistical queries could be revealed to another database owner. Such technique can be used to prevent PPRL techniques from releasing information which would identify individuals in the databases [30, 99].

2. **Secure multi-party computations (SMC) based methods**
Secure multi-party computation has been introduced as a solution to overcome the problem of performing computations on sensitive data across organisations. SMC enables several parties to be involved in a computation with their private input data, where at the end of the computation no party learns anything about any other parties' private data except the final result of the computation. Besides PPRL [53, 189, 213], SMC has been used in application areas including privacy-preserving data mining [45] and privacy-preserving data publishing [79].

   In 1986 Yao [233] introduced the idea of secure computation between two data owners which was extended by Goldrich et al. [84] in 1987 for multiple data owners. Under SMC protocols, a variety of encryption schemes, such as homomorphic encryption [161], and techniques such as secure summation, secure set union, secure set intersection, and secure scalar product, have been proposed.

   These techniques are used in PPRL for accurate computation while preserving privacy [45, 53, 118, 176, 213]. Some of the techniques that can be categorised under SMC are:

   - **One-way hash encoding**
   One-way hash encoding has been used in PPRL due to its property of converting a attribute value into a hash code [21, 190, 214]. Some well known one-way hash algorithms such as the message digest (for example, MD5) [128] and secure hash algorithms (SHA-1 and SHA-2) [128, 187] are commonly used to encode attribute values. Conversion of an attribute value into a one-way hash code makes it difficult for an adversary to learn the original attribute value from its hash code [187]. A major drawback of using hash encoding for matching records between databases is that errors or variations in attribute values will lead to different hash codes [21]. This becomes problematic with real-world data as these can contain errors, variations, missing values, and values that can be out of date [95].

- **Oblivious transfer protocols**
  Oblivious transfer (OT) protocols [168] have been used in several PPRL techniques [88, 108, 160, 224] due to their capability of exchanging information among parties that participate in a linkage protocol while preserving privacy. OT protocols allow a sender to send parts of its input to a receiver in such a manner that protects the sensitive input data of both the sender and receiver by ensuring that the sender does not know which part of its data is received by the receiver, and the receiver does not learn any information about the other parts of the sender's input. In general, OT protocols are computationally and communicationally expensive, and often become the efficiency bottleneck in protocol design [88, 100].

- **Homomorphic encryption**
  Homomorphic encryption (HE) schemes allow computations between data to be performed in an encrypted form (ciphertext) and output an encrypted result [81, 118, 213]. HE ensures that the decrypted result matches the result of the same function performed with the unencrypted data. For example, assume two given numbers n_1 and n_2 belong to data owners Alice and Bob, respectively. First, Alice and Bob encrypt their numbers n_1 and n_2 into an encrypted form ε_1 and ε_2, respectively, using an encryption function $E()$, where $\varepsilon_1 \leftarrow E(n_1)$ and $\varepsilon_2 \leftarrow E(n_2)$. Based on a homomorphic addition scheme, Alice and Bob can compute the summation of ε_1 and ε_2, denoted as $(\varepsilon_1 + \varepsilon_2)$. The decryption of $(\varepsilon_1 + \varepsilon_2)$ equals to the addition of the plain counterparts n_1 and n_2, denoted as $(n_1 + n_2) = D(\varepsilon_1 + \varepsilon_2)$, where $D()$ is a decryption function.

  Homomorphic encryption can be categorised into fully and partially (somewhat) homomorphic schemes [154]. Fully homomorphic schemes can perform addition and multiplication [161] or any arbitrary calculation [81]. However, these schemes are computationally not efficient due to their complex encryption and decryption operations. Partially homomorphic encryption schemes only support a limited number of operations on encrypted data, however, they are much faster and thus more practical [154].

  In homomorphic encryption schemes a key pair, known as public and private (secret), is used to encrypt and decrypt the private input data accordingly [161]. The public key is kept publicly available to any party that participates in a protocol while private key is not shared and kept secret by the party who generate the key pair. Successive encryption of the same value using the same public key generates different encrypted values with high probability, and decrypting the encrypted values using a private key returns the correct original value. Homomorphic encryption has been used in PPRL to compute similarities between private input data held by different data sources [76, 107, 118, 175, 213].

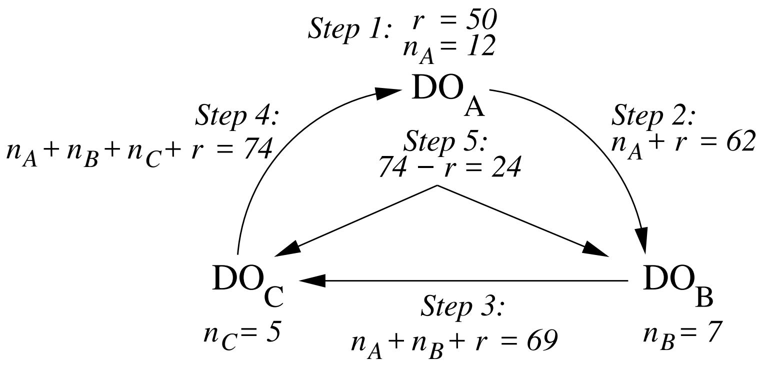


Figure 2.2: An example of secure summation protocol between database owners DO_A, DO_B, and DO_C to compute their private input values n_A, n_B, and n_C, respectively. In this example, DO_A initialises the protocol by generating a random number r (step 1) and adds r to its private input n_A (step 2). In steps 3 and 4, DO_B and DO_C, respectively, updates the partial sum it received from previous DO by adding its own private input value and sends the updated partial sum value to the next DO. Finally, in step 5 DO_A sends the final sum to other DOs by subtracting the r from the partial sum value it received from DO_C.

- **Secure summation protocol**
  Secure summation is an SMC protocol that has been used in several PPRL approaches [45, 140, 160]. This protocol allows multiple cooperating DOs (more than two) to compute the sum of their individual input values (assumed to be numbers) without revealing their data to the other parties participating in the protocol.

  By assuming three DOs, DO_A, DO_B, and DO_C, the idea behind the secure summation protocol for can be described as follows [45]. Let us assume private numerical values, n_A, n_B, and n_C held by three DOs, DO_A, DO_B, and DO_C, respectively. These DOs want to compute the summation of their numbers. Initially DO_A chooses a large random number r and then adds r to its input n_A. Then DO_A sends this partially summed value to DO_B. Since r is random, DO_B learns effectively nothing about n_A.

  DO_B adds its private input number n_B to $(r + n_A)$, and sends the result $(r + n_A + n_B)$ to DO_C. Following the same steps of DO_B, DO_C adds its n_C to the partial sum it received from DO_B, and the newly computed partial sum $(r + n_A + n_B + n_C)$ is sent to DO_A. Then DO_A subtracts r from the final partial sum value and the resulting sum $(n_A + n_B + n_C)$ is sent to DO_B and DO_C. Once the computation is finished each DO only knows the total sum, from which they are unable to derive the other DOs' numbers. Figure 2.2 illustrates an example of computing the summation of numerical values, n_A, n_B, and n_C held by DOs, DO_A, DO_B, and DO_C, respectively.

  As we will discuss in Chapters 5 and 6, we use secure summation protocol as a building block in our blocking approaches. However, the secure summation protocol described above is susceptible to different privacy risks (as will be discussed in Section 2.4.5), especially collusion, as DOs can share with each other their own data and partially computed results they receive from other DOs to compute an input for a DO that does not collude. In Section 4.3.3, we analyse existing secure summation protocols in terms of their privacy risk and we propose a novel secure summation protocol that addresses these privacy risks.

- **Secure set intersection**
  Secure set intersection (SSI) protocols have been used in PPRL approaches to compute the intersection of sets of records in different databases without revealing any additional information about the values that are not common in those databases [3, 53, 224]. SSI protocols generally use either commutative [2] or homomorphic [148] encryption schemes. Several efficient SSI protocols have been introduced with linear communication complexity [224]. However, these protocols are computationally expensive since they are based on complex encryption algorithms, which therefore need to be scalable by running computations in parallel [53]. Furthermore, SSI protocols can be used in PPRL scenarios where a trusted third party is not available to carry out the linkage [125, 205, 206].

2.4.5 Privacy Attacks

As we will describe in Chapter 3, over the past years various linkage techniques, that are based on different perturbation and SMC methods described above, have been proposed for PPRL. Though perturbation techniques are more efficient compared to SMC techniques, these techniques are vulnerable to various privacy attacks due to the presence of partially revealed information [212]. The main attacks and vulnerabilities of PPRL techniques are:

1. **Frequency attack**
   Frequency attacks are based on the frequency distribution of a set of encoded (masked) values as compared to the frequency distribution of a set of known plain-text values. Even without knowing the parameters used in the encoding function, the encoded values can be re-identified by matching their frequencies with the frequency distribution of plain-text values in a known global database [39, 141]. An attacker does not require any prior knowledge about the encoding process used in the masking of attribute values, but only requires knowledge on which attributes are encoded and have access to a public database of these attribute values and their frequencies from the same domain as the encoded database [39].

2. **Dictionary attack**
   Dictionary attacks require an adversary to know about the encoding function and some of the parameter values used in a PPRL protocol [212]. Similar to frequency attacks, an attacker uses a publicly available database to analyse the

encoded attribute values. To identify encoded attribute values, the attacker applies the same encoding function and parameter values to encode the attribute values in the public database until a matching encoded value is found. Dictionary attacks can be overcome by using secret key (or hashed key [128]) based encoding functions which makes a dictionary attack harder without knowing the correct secret key.

3. **Composition attack**
   In composition attacks, an adversary needs to have background knowledge (auxiliary information) about the individual databases that are to be linked and/or certain records of these databases. Using this auxiliary information from different databases that have been linked, an attacker could learn the distances or similarities between sensitive attribute values of a given record [80, 138]. As an example of such an attack, assume two hospitals in the same city release anonymised patient-discharge information. Because they are in the same city, some patients may visit both hospitals with similar illnesses. By using the overlapping patient population in these anonymised databases, an attacker could identify an illness of a person he knows and who lives in this city.

   Known schema attack is a type of composition attack that can also be applied in a PPRL context, where an attacker knows the details of the encoding methods that are used by the database owners and tries to break the encoding scheme by utilising the knowledge of the encoding methods [52].

4. **Cryptanalysis attack**
   A cryptanalysis attack is a sub-category of a frequency attack, where an attacker uses a set of encoded values and a publicly available database to identify the sensitive attribute values by conducting a frequency analysis based on the encoded values [132]. In PPRL, Bloom filter based techniques are vulnerable to cryptanalysis attacks [39, 56, 190]. Because of the distribution of 0 and 1 bits in Bloom filters, an attacker can potentially learn the characteristics of the hash functions that were used to map attribute values into these Bloom filters.

   Existing cryptanalysis attacks are feasible only based on assumptions made about the parameter settings used in the Bloom filter encoding phase and they also require excessive computational resources making them not practical in real settings [130, 132, 133, 158]. However, a recent cryptanalysis attack by Christen et al. is capable of re-identifying encoded attribute values in Bloom filters based on the construction principle of how q-grams are hashed into Bloom filters [39]. This attack is independent of the encoding function and its parameters used, and computationally inexpensive.

5. **Collusion**
   In multi-party protocols, collusion can occur between the participating parties [44, 160, 212]. Recently in different application domains, such as online rating, auctioning, and mobile computing, many data related applications involving multiple parties have experienced collusive behaviour in their partici-

pating parties that are trying to gain access to unauthorised data [4, 20, 50, 202]. The aim of collusion is to learn the sensitive data of another not colluding party by the colluding parties sharing their own data and parameter settings [3, 219]. This requires any application that is used in a semi-honest or malicious context to be collusion resistant [5, 140]. Different types of collusion scenarios might occur with regard to the linkage model employed in MD-PPRL as one or several DOs collude with the linkage unit, or alternatively a subgroup of DOs collude to learn about other DOs' sensitive data. We will discuss these different collusion scenarios in detail in Section 4.3.

2.5 Evaluation Measures

The final step of a linkage process is to measure the performance of the linkage conducted. In RL the performance is measured by evaluating the efficiency and effectiveness of the techniques used, while in PPRL privacy is also measured as another evaluation criterion. We discuss different measures that are used for both RL and PPRL for evaluating scalability and linkage quality, and privacy measures that are used in a PPRL context.

2.5.1 Efficiency (Scalability) Evaluation

Efficiency measures the scalability of a technique used in the linkage process. Scalability can be measured using metrics based on the computational environment or the number of candidate record pairs generated by an algorithm for comparison. The measures that can be used based on the computational environment are:

1. **Runtime**
   This measure is based on the time requirements to produce the results of a linkage. It can include the time required for the different steps in the linkage process, including pre-processing, blocking, comparison, and classification.

2. **Communication Usage**
   In a linkage across several organisations, it is important to measure how much data (total and average message sizes) are communicated between the participating parties. The communication usage is measured as the number of messages and their sizes (in bytes). Also, the total number of messages communicated between the participating parties or the average number of messages sent by a party can also be used to measure the amount of communication usage of a linkage protocol.

3. **Memory Usage**
   For computations, the data need to be read from secondary memory into main memory for easy and fast accessibility. This metric measures the total main memory usage by an algorithm when performing the linkage.

In RL and PPRL, scalability of blocking can also be measured based on the number of record pairs generated by the blocking technique [36].

1. **Reduction Ratio (RR)**
   Reduction ratio is a commonly used scalability measure in RL which provides information about the size of the comparison space generated by a blocking technique. RR requires two metrics, (1) the number of candidate record pairs generated by blocking and (2) the number of all possible record pairs that can be generated across the databases. Based on these two metrics the reduction ratio can be calculated as:

$$RR = 1.0 - \frac{number\ of\ candidate\ record\ pairs}{number\ of\ all\ possible\ record\ pairs}$$

We will define an adapted reduction ratio measure for a MD-PPRL protocol in Section 4.4.

2.5.2 Effectiveness (Quality) Evaluation

Effectiveness measures the accuracy of a linkage process. In a RL or PPRL protocol the quality of blocking and classification can be measured separately. In order to measure the effectiveness for linkage of two databases, the values of the number of true match (T_M) and true non-match (T_N) record pairs identified by the blocking algorithm, and the total number of true match (N_M) and true non-match (N_N) record pairs available across these two databases need to be counted. The blocking quality can be measured as follows:

1. **Pairs Completeness (PC)**
   Pairs completeness measures the effectiveness of grouping the true matching records into blocks [36]. A lower pairs completeness value indicates that more true matching record pairs were removed by a blocking technique from the comparison space. Pairs completeness is similar to the recall measure as used in information retrieval [229] and is defined as:

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

2. **Pairs Quality (PQ)**
   Pairs quality is another metric that measures the quality of the blocking step in the linkage process [36]. It assesses the number of actual true matching record pairs included in the candidate record pairs generated by a blocking technique. Pairs quality is computed as:

$$PQ = \frac{T_M}{T_M + T_N}$$

The accuracy of the classification model can be measured by evaluating the number of true match (C_M) and true non-match (C_N) record pairs identified by a classifier.

1. **Precision**
   Precision is commonly used as a measure in information retrieval to measure the quality of a search (query) result [36, 229]. In RL and PPRL precision provides an assessment about the number of true matching record pairs classified as matches by a classification model used in the linkage process. This measure is similar to pairs quality. In RL and PPRL precision is calculated as:

$$Precision = \frac{C_M}{C_M + C_N}$$

2. **Recall**
   In RL and PPRL recall is used to measure the number of true matches correctly classified by the classification model compared to the total number of true matches available across the databases [36, 229]. This measure is also known as *sensitivity* [136]. Similar to PC, recall is measured as:

$$Recall = \frac{C_M}{N_M}$$

3. **F-measure/ F-Score**
   The F-measure is often used in information retrieval to measure the accuracy and performance of a binary classification technique [229]. The F-measure is the harmonic mean of precision and recall. Based on the precision and recall measures given above the F-measure (*FM*) can be calculated as:

$$FM = 2 \times \left(\frac{Precision \times Recall}{Precision + Recall} \right)$$

Recently, Hand and Christen showed that the F-measure can be alternatively calculated by using a weighted arithmetic mean between precision and recall [92]. The weights are used to assign a relative importance to precision and recall. These weights are computed based on the number of predicted matched record pairs and the similarity threshold used in the classification technique. The selection of these weights is important in comparative studies when different classification or linkage techniques are to be evaluated equally. To assign the same weights for precision and recall for all methods, each method must classify the same number of record pairs as matches. To accomplish this Hand and Christen suggested to use different choices of the similarity threshold for different methods so they classify the same number of record pairs as matches. Furthermore they showed that the computation of these weights depends on the linkage problem and should be chosen by the researcher or user, and it would be beneficial for future studies to report these weight choices in any comparative study.

Several other measures such as accuracy, error rate, and specificity can also be used to measure the quality of the classification model used in the linkage process. However, these measures are not suitable because the classification step generally is a class imbalance problem as many more record pairs are classified as non-matches compared to matches [38]. The number of true non-matches can significantly bias the computed values of these measures. Precision, recall and F-measure are the most commonly used measures for assessing the quality of the classification in RL [35].

2.5.3 Privacy Evaluation

Privacy is a crucial aspect in any step of PPRL. Several standard information theory measures such as *entropy*, *information gain* (IG), and *relative information gain* (RIG) have been used in PPRL to assess the privacy of deducing a record in a database [56, 112] using a publicly available global database. These measures are calculated based on a simulation attack on the encoded (masked) database ($\mathbf{D}^M$) using the original database $\mathbf{D}$ such that these measures assess the possibility of inferring records in the original database given its encoded version.

In information theory entropy measures the amount of information contained in a message [197]. Entropy is a function of the probability distribution over the set of all possible messages. The entropy (H) of a message X with x being an element in X is defined as:

$$H(X) = -\sum_{x \in X} p(x) \cdot log_2\, p(x).$$

Following Durham [56] and Karaksidis et al. [112], in PPRL the entropy of a database $\mathbf{D}$ can be calculated as:

$$H(\mathbf{D}) = -\sum_{R \in \mathbf{D}} (n/|\mathbf{D}^M|) \cdot log_2\, (n/|\mathbf{D}^M|),$$

where n denotes the number of records in $\mathbf{D}^M$ that match with a record R in $\mathbf{D}$, and $|\mathbf{D}^M|$ is the total number of records in $\mathbf{D}^M$.

Information Gain (IG) is closely related to the entropy measure. IG is a metric which can be used to assess the difficulty of inferring the original database ($\mathbf{D}$), having only its encoded version ($\mathbf{D}^M$), or how the knowledge of $\mathbf{D}^M$ can reduce the uncertainty of inferring $\mathbf{D}$ [56, 110]. IG between $\mathbf{D}$ and $\mathbf{D}^M$ is defined as:

$$IG(\mathbf{D}|\mathbf{D}^M) = H(\mathbf{D}) - H(\mathbf{D}|\mathbf{D}^M),$$

with $H(\mathbf{D}|\mathbf{D}^M)$ being the conditional entropy between $\mathbf{D}$ and $\mathbf{D}^M$. Lower IG means that it is more difficult to infer $\mathbf{D}$ from $\mathbf{D}^M$. Relative information gain (RIG) provides a normalised scale with regard to the entropy of $\mathbf{D}$ and is defined as:

$$RIG(\mathbf{D}|\mathbf{D}^M) = \frac{IG(\mathbf{D}|\mathbf{D}^M)}{H(\mathbf{D})}$$

However, according to Vatsalan et al. [212], a standard and normalised set of measures are required to quantify privacy based on simulated attacks [134]. Vatsalan et al. [216] introduced a set of disclosure risk (DR) measures that can be used to evaluate and compare different private blocking and classification solutions. The assessment of DR uses a measure of re-identifiability of a record in $\mathbf{D}^M$ by using an external global database ($\mathbb{G}$) which is practically perform by conducting re-identification studies by linking values from $\mathbf{D}^M$ to ($\mathbb{G}$) [54]. This is measured by calculating the probability of suspicion (PS) value which provides how many records in $\mathbb{G}$ can be linked with a corresponding record in $\mathbf{D}^M$. This measure of PS is normalised into 0 and 1, where 1 indicates that a record in $\mathbf{D}^M$ can be exactly re-identified with a record in $\mathbb{G}$ based on one-to-one matching, and 0 means a record in $\mathbf{D}^M$ could correspond to any record in $\mathbb{G}$ and therefore it cannot be re-identified.

If n_g is the number of records in a global database $\mathbb{G}$ that match to a record R in the encoded database $\mathbf{D}^M$ then the normalised PS of R is calculated as:

$$PS(R) = \frac{\frac{1}{n_g} - \frac{1}{|\mathbb{G}|}}{1 - \frac{1}{|\mathbb{G}|}},$$

where $|\mathbb{G}|$ represents the number of records in $\mathbb{G}$. Based on the normalised PS values for each record in $\mathbf{D}^M$ several statistical DR measures can be defined [212].

1. **Maximum disclosure risk (DR_{Max})**
   This measures the maximum PS for a record in $\mathbf{D}^M$ which can be computed as:

   $$DR_{Max} = \max_{R \in \mathbf{D}^M} (PS(R))$$

2. **Mean disclosure risk (DR_{Mean})**
   This measures the average of PS values to evaluate the average disclosure risk which is:

   $$DR_{Mean} = \frac{\sum_{R \in \mathbf{D}^M} (PS(R))}{|\mathbf{D}^M|}$$

   where $|\mathbf{D}^M|$ corresponds to the total number of records in $\mathbf{D}^M$.

3. **Median disclosure risk (DR_{Med})**
   The median DR gives the center of the distribution of DR values by taking into account the distribution of PS values in the encoded database $\mathbf{D}^M$. By assuming the PS values of records in $\mathbf{D}^M$ are sorted from the lowest to the highest, the median disclosure risk can be calculated as:

   $$DR_{Med} = \begin{cases} \frac{PS(R_{\frac{|\mathbf{D}^M|}{2}}) + PS(R_{\frac{|\mathbf{D}^M|+1}{2}})}{2} \; and \; R_{\frac{|\mathbf{D}^M|}{2}}, \; R_{\frac{|\mathbf{D}^M|+1}{2}} \in \mathbf{D}^M & \text{if } |\mathbf{D}^M| \text{ is even} \\ PS(R_{\frac{|\mathbf{D}^M|}{2}}) \; and \; R_{\frac{|\mathbf{D}^M|}{2}} \in \mathbf{D}^M & \text{if } |\mathbf{D}^M| \text{ is odd} \end{cases}$$

4. **Marketer disclosure risk (DR_{Mkt})**
   This measure defines how many values in the encoded database can be exactly re-identified where their PS value is 1.

$$DR_{Mkt} = \frac{|\{R|PS(R) = 1 \text{ and } R \in \mathbf{D}^M\}|}{|\mathbf{D}^M|}.$$

2.6 Summary

In this chapter, we have provided the background on the concepts, principles, and techniques which lay the foundation for the contributions in this thesis. We covered traditional record linkage (RL) followed by a detailed description of the privacy-preserving record linkage (PPRL) process. We described different privacy aspects in PPRL and finally presented the evaluation measures used in both RL and PPRL. In the next chapter, we will survey the related relevant work in PPRL. We will use these surveyed research papers as inspiration and guidance for the novel contributions of this thesis in the later chapters.

Chapter 3

Current Progress in Privacy-Preserving Record Linkage

This chapter provides a detailed description of the current progress in the PPRL research area. Details of different algorithms and techniques developed in PPRL are provided for the blocking and classification steps of the linkage process. For each step we categorise the techniques based on the number of databases to be linked and each category is ordered according to the year of publication.

3.1 Blocking Techniques for PPRL

Blocking techniques are employed to facilitate a linkage to scale to very large databases that contain millions of records. In PPRL, blocking becomes challenging because of privacy concerns. This introduces a trade-off in the blocking step not only between accuracy and efficiency, but also privacy.

Recently, a variety of blocking approaches have been developed for reducing candidate record pairs (or sets) that need to be compared in record linkage. In the surveys by Papadakis et al. [165] and Christen [36] comprehensive reviews about existing blocking mechanisms are provided. Some of the developed approaches have been adapted for PPRL [56, 109, 118, 134, 216] based on existing blocking techniques, such as standard blocking [71], mapping based blocking [110], clustering [48, 134, 144, 215], and locality sensitive hash (LSH) functions [56, 101].

3.1.1 Two Database Protocols with a Linkage Unit

- Al-Lawati et al. [3] introduced a blocking mechanism for PPRL which increases the performance of the overall linkage process. They based their approach on the linkage model with a *linkage unit* (LU) under the *honest-but-curious* (HBC) adversary model. A trusted third party (TTP) is used as the LU to do the comparison of the records sent to it by the database owners.

  In their approach three blocking methods have been developed to reduce the complexity of the linkage process. They are simple blocking, record-aware blocking, and frugal third-party blocking mechanisms. In simple blocking, hash signatures of records are assigned to blocks. However, in simple blocking the similarity of a record pair may be computed multiple times if records are assigned to multiple blocks. To overcome this problem the authors introduced a record-aware blocking mechanism which can reduce this overhead by combining an identifier with the hash signature of each record which provides a higher reduction ratio than the simple blocking mechanism. The frugal third-party blocking uses a secure set intersection (SSI) SMC protocol to reduce the cost of transferring the whole databases to the LU by first identifying the hash signatures that occur in both databases.

In this approach the record pairs are classified using a threshold based classification model. The LU matches the TF-IDF distances of the hash signatures to measure the matching of candidate record pairs over a given threshold value using the *Jaccard* similarity metric [35]. The Jaccard similarity sim_J between two hash signatures of h_A and h_B is calculated as,

$$sim_J(h_A, h_B) = \frac{|h_A \cap h_B|}{|h_A \cup h_B|}, \tag{3.1}$$

where $|h_A \cap h_B|$ and $|h_A \cup h_B|$ are defined as the cardinality (size) of the intersection and union of hash signatures h_A and h_B, respectively.

- Inan et al. [98] proposed a hybrid approach that combines value generalisation and cryptographic techniques for PPRL. Their blocking approach uses a value generalisation hierarchy on the blocking attributes. The records are assigned to the corresponding block in the value hierarchy based on the blocking attribute values. However, the record pairs that cannot be blocked are compared in a computationally expensive *secure multi-party computation* (SMC) approach using cryptographic techniques. Experiments conducted on real datasets show that their blocking approach can significantly reduce the number of candidate record pairs that need to be compared in the SMC step and yields much more accurate matching results compared to sanitisation techniques, even when the datasets are perturbed extensively. However, their method is useful only with attributes that can form hierarchies.

- Karakasidis and Verykios [111] introduced a blocking approach for PPRL which uses a LU with the HBC adversary model. The authors used a reference value based blocking mechanism that exhibits better performance than other reference value based methods [110, 112, 186, 190] available in the PPRL context. The proposed approach uses the nearest neighbour clustering algorithm for forming initial clusters using a common set of reference values. The database owners need to have an agreement on the set of attributes which will be used for the blocking and matching of record pairs. The reference values are selected for each cluster ensuring that all the clusters contain at least k reference values to guarantee k-anonymous privacy.

The set of blocks are created by each database owner (DO) by assigning each record in their database to the respective block based on the similarity between the attribute values and the reference values in the clusters. The *Dice-coefficient* (Dice similarity) [35] is used to assess the similarity between these reference values and attribute values. Dice similarity (sim_D) for two sets A and B is calculated as follows,

$$sim_D(A, B) = \frac{2 \cdot |A \cap B|}{|A| + |B|}. \tag{3.2}$$

After blocking is performed each DO sends their blocks to the LU. The LU merges the corresponding blocks from two DOs to generate candidate record pairs for comparison. The empirical study conducted on synthetic datasets has shown that the proposed approach performs accurately and provides high robustness to frequency attacks but with higher computational time requirements than the simple blocking approach proposed by Al-Lawati et al. [3].

- Karakasidis and Verykios [113] then introduced another blocking technique for PPRL based on sorted neighbourhood clustering [95]. The proposed approach uses reference values to generate initial clusters similar to their previous blocking approach [111]. However, in the suggested approach the authors use the k-medoid clustering technique to assign reference values into clusters. Both DOs need to agree on the same parameter settings to ensure the same clustering is generated for each database.

  Once the clusters are generated each DO assigns the records in its database by calculating the edit distance between each blocking attribute value of each record and the clusters of the clusterings. Based on the calculated distance a record might be assigned to several clusters. Each DO performs these steps independently to generate their clusters.

  These generated clusters are then sent to the LU. The LU merges the corresponding clusters sent by the DOs and includes the records from each pair of clusters into a single adjacency list. Then the records in each of these lists are sorted by a score calculated by the sum of their distances from the respective clusters.

  Finally, the LU applies a sorted neighbourhood approach on these sorted lists to generate candidate record pairs for comparison. A fixed size window is moved over each of the adjacency lists and those records from two databases falling within this window are paired together for comparison.

- In 2012, Durham et al. [56] proposed a Bloom filter (BF) based method for private blocking and classification based on record level Bloom filter (RBF) [59] encodings. RBFs are made from attribute level Bloom filters of dynamic size by a weighted sampling of random bits as measured by the discriminatory power of different attributes.

According to the authors their blocking technique has three main steps, which are:

- Blocking variable identification
  In the first step RBFs are generated for blocking attributes of each record in the database. The suggested RBF encodings can be used to overcome the problem of cryptanalysis attacks associated with attribute level Bloom filter encodings.
- Spatial partitioning
  The term *spatial partitioning* is particularly 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. Once the blocking variable is selected, it will be used to divide the set of records into different partitions. A Hamming distance based locality-sensitive hashing (HLSH) approach is used for spatial partitioning of RBFs by selecting a group of dimensions or bit positions in a Bloom filter for each block. RBFs which have the same values in these bit positions are added to the same block.
- Record pair generation
  Record pairs are generated from all the records in the same blocks.

This approach uses a class of LSH functions that approximate Hamming distance based on bit sampling. HLSH accepts RBF encodings corresponding to a set of records as input, and generates the list of record pairs that need to be considered in the classification step of linkage. The performance of HLSH is varying depending upon the number of hash functions used and the number of iterations used in the block generation.

Durham et al. argued that the application of HLSH on RBF encodings provides the relevant privacy guarantee on the databases where an attacker could not use the blocks to determine information about the attribute values. Also, an attacker cannot infer knowledge about the attribute values through the frequency distribution of the bits in RBFs because they are generated by randomly selecting bits from individual attribute level BFs. Empirical studies conducted on real databases showed that HLSH performed better than the token based blocking approach [35] while achieving higher accuracy in the linkage and faster runtime.

- In 2013, Vatsalan and Christen [209] also utilise a sorted neighbourhood clustering approach for blocking in PPRL. The authors also use reference values to form initial blocks which are shared between the two DOs. In this approach each DO sorts the set of reference values and then inserts its records into the sorted list according to their sorting key. In the initial cluster generation each reference value is assigned to a single cluster and records are added to these clusters. To ensure k-anonymity, these initial clusters are merged into larger blocks containing at least k records. For merging of clusters the authors suggested two techniques based on similarity or size constraints. Once the k-anonymous blocks are generated each DO sends their blocks with encoded records to the LU for comparison which follows similar as [111].

- The concept of using LSH functions in PPRL for blocking has been extended further by Karapiperis and Verykios [118]. The authors presented a Λ-fold redundant blocking framework that relies on LSH for identifying candidate record pairs that need to be compared. The proposed approach uses a LU and assumes the HBC adversary model. The proposed approach can be employed with three families of hash functions, namely the Hamming family (as used by Durham et al. [56]), the Min-Hash family and the ρ-stable distributions-based family. Before the records are grouped into blocks each DO anonymises their records by using either Bloom filters or an embedding method in the Euclidean space.

  The LU generates the required set of hash functions based on an agreed hash family. This set of hash functions is sent to the DOs to hash the anonymised records and populate blocks accordingly. The matching of the record pairs in these blocks can be implemented by performing simple distance computations or by a more sophisticated SMC-based protocol based on homomorphic distance computations. The experimental results indicated that LSH based blocking approaches can significantly reduce the number of candidate pairs that need to be compared.

  The same authors [117] recently extended this blocking approach into a distributed framework which relies on Apache Hadoop based MapReduce framework[1] in order to distribute computations among under-utilised commodity hardware resources uniformly, without imposing extra overhead on the existing infrastructure. In this approach, records are encoded into Bloom filters in order to protect the privacy of the underlying data, and then they are submitted to the LU that splits and distributes these Bloom filters to a file system with replication capabilities to improve scalability. The Min-Hash LSH technique [25] is applied on each Bloom filter to arrange Bloom filters into blocks. The evaluation of this parallel LSH approach was limited to small datasets and distributed computer nodes which makes the overall scalability of the approach unclear.

- Cao et al. [30] proposed a private record linkage framework based on differential privacy (as was described in page 28). The main idea of this approach is to allow database owners to partition their databases into non-overlapping subsets independently and release a synopsis of each subset to a LU to identify similar subsets of records that need to be compared by measuring the distance between those synopsis. In this approach the LU's knowledge on records is kept to a minimal value by generating all these synopsis differentially private.

[1]http://hadoop.apache.org/

During the database partitioning step, each DO follows a hierarchical approach to partition its database forming a tree. It starts from a single node (i.e., the root) that covers all the records in a database. Then, it recursively partitions the nodes. Whenever a node in the tree needs to be split, it dynamically allocates a privacy budget based on the node size. The privacy budget is used to check whether to split nodes and if the leaves contain appropriate number of records to ensure privacy. Finally, the subsets of records in each leaf is sent to the LU to identify the matching records. The experimental results showed that this approach is more efficient and accurate compared to a previous differentially private linkage approach [99].

- The use of reference values for blocking in PPRL has recently been further extended by Karakasidis et al. [109]. They proposed a blocking technique that groups records into blocks by comparing them against a reference set. The aim of this approach is to use multiple reference set samples which can assign a record into multiple blocks. The multiple assignments of a record in blocks introduce redundancy in the blocking process. This differs from [111, 113, 209] which use a single reference set as the comparison basis.

  In the suggested approach, both DOs need to agree on the parameters for generating identical reference sets. The DOs share a reference value database where they sample sets of reference values using the Durstenfeld shuffle algorithm [60]. Each DO then independently assigns the records in their databases to blocks based on the edit distance between the reference values and the attribute values. To improve the redundancy of blocking several blocking key attributes are used to form blocks.

  Once the blocking is finished, each DO sends their blocks with record identifiers to a LU to identify candidate record pairs for comparison. The LU merges the corresponding blocks from each DO to generate candidate record pairs and finally the DOs can send their encrypted record values to the LU for these generated candidate pairs to compare using any private matching method [216]. Empirical evaluations with real databases showed that the suggested approach can achieve high linkage quality and link dirty data due to the redundancy in blocking based on the use of multiple reference set samples and multiple record to block assignments.

- Karapiperis and Verykios [119] recently proposed a blocking scheme by using Hamming locality-sensitive hashing (HLSH) technique as done by Durham et al. [56]. The main idea behind this approach is to encode records into the Hamming metric space by using the Bloom filter encoding [18] and assign these Bloom filters into independent blocking groups by applying the HLSH technique.

  First each DO encodes all the records in their databases into Bloom filters (BFs) based on an agreed set of encoding parameters. These generated BFs are then sent to a LU. The LU then uses a HLSH based blocking technique to block the received BFs and formulate BF pairs for comparison. In the blocking process record pairs are hashed into several hash buckets forming redundant blocks.

To identify the pairs to be compared the LU counts the number of collisions (number of times a given pair of BFs is hashed into the same hash bucket). The LU calculates the minimum number of collisions required by a BF pair to consider it to be compared and then performs the Hamming distance computations only for these pairs that exhibit this minimum number of collisions. Empirical results showed significant improvement in running time due to a drastic reduction of candidate pairs by the HLSH technique, while achieving high blocking quality.

- Recently, Han et al. [90] proposed a private blocking approach for PPRL based on dynamic k-anonymous blocking and the Paillier cryptosystem. The proposed approach can only be used on numerical values since the suggested k-anonymous blocking dynamically generates blocks by using numerical values as reference values to assign k records into blocks for satisfying k-anonymity. However, the suggested approach is different from the approach by Vatsalan and Christen [209], since the suggested approach does not use any publicly available reference dataset to generate blocks, but uses the attribute values in the database as reference values.

  First the DOs agree on the parameter settings, including blocking key attributes and minimum k to be used in the block generation process. Each DO then forms blocks on their database ensuring each value of blocking key attribute constructs one block. To ensure k-anonymity, similar blocks are merged together until the number of records in a block being at least k.

  Once the blocks are formed locally, each DO computes reference values for each block. The smallest and the largest values in a block are selected as the reference values for each given block. To identify similar blocks that need to be compared one DO needs to release its reference values to the other party. Before sending these reference values each database owner encrypts these values by using a public key sent to them by a LU.

  Once the reference values are encrypted one DO sends these values to the other DO to compute the similarity by using the Paillier cryptosystem [161]. The computed encrypted similarity values are sent back to the LU to decrypt by using the private key. The decrypted results are then sent back to the DOs where based on the received similarity values they decide whether corresponding blocks need to be compared or not. Experiments with a real dataset showed that the suggested approach can achieve higher accuracy and privacy, but does not scale to database with large number of records.

- Recently, Chi et al. [31] proposed a technique for linking two databases in the presence of missing values. Their approach uses the attribute values of the nearest neighbours of a record to impute missing values. The proposed approach uses a Bloom filter based traditional blocking technique to improve the scalability, but requires a quadratic computation complexity in the missing value imputation process. Nevertheless, for the imputation process, the authors assumed that the record with a missing value and its nearest neighbour records have similar values for the corresponding attribute. However, such an assumption is not always realistic in real-world situations. For example, the people who live at the same address can share different last name or telephone numbers. Hence, as we discuss in Section 11.4, an extension of such an approach to MD-PPRL requires further research because such techniques need to scale to an increasing numbers and sizes of databases.

3.1.2 Two Database Protocols without a Linkage Unit

- A protocol proposed by Song et al. [198] tried to address the problem of approximate matching by calculating enciphered permutations for private approximate record matching. The proposed technique provides remote searching on encrypted data using an untrusted server and provides proofs of security for the resulting crypto systems. The encryption and search algorithms only need $O(n_r)$ stream and block cipher operations for a database with n_r records.

  A query can either be a single word or multiple words which is/are sent to the server to perform a search. If an encrypted word matches at least one of the enciphered permutations, then the pair of words can be considered as a match. The use of an encrypted block data structure provides an efficient search even when the document size is large. However, the approach is vulnerable to frequency attacks if a certain number of words are being queried, and therefore not suitable for real-world applications.

- Inan et al. [99] proposed a novel blocking mechanism which is based on differential privacy. Records in a database are partitioned into subsets with the aim of reducing the cost of private record linkage. The approach uses specialised multi-dimensional tree index data structure to improve the scalability of the partitioning step. Based on a given distance function, values in each partition are evaluated by a decision rule. The pairs that are considered as matches by the decision rule will be classified by a SMC technique. Experimental results showed that differential privacy provides strong privacy guarantees and a trade-off between accuracy, privacy, and scalability.

- Karakasidis and Verykios [110] proposed a complete framework for PPRL for two database scenarios. The authors enhanced edit distance based methods used for conventional approximate record matching with privacy-preserving characteristics and increased the matching performance by facilitating a secure blocking component.

  In this approach phonetic encoding was used as a blocking component, which matches words based on their pronunciation and has fault tolerance against typographical errors. The authors used the Soundex [35] and Metaphone [35] algorithms to perform the phonetic encodings and the MD5 hashing algorithm to encrypt the phonetic encodings into ciphers. The phonetic encodings were injected with a random number of records consisting of fake phonetic codes to ensure that the phonetic values exhibit uniform distributions. Identical ciphers are grouped together for matching.

In their proposed approach, for each of the generated groups of encrypted phonetic encodings a secure matching mechanism was used to perform approximate matching between the records. For the value held by each field in the record each of its characters are extracted. These characters are appended with a number to indicate its position which results in a set of tokens. Every token is then encoded into a Bloom filter. The set of Bloom filters are then exchanged to perform the matching process. Recently, Etienne et al. [68] also evaluated the use of phonetic codes in private blocking techniques which showed how errors and modifications in string values affect the quality of blocking.

- The applicability of multi-bit tree data structures as a blocking mechanism was explored by Schnell [188] for PPRL. The multi-bit tree data structure was proposed by Kristensen et al. [129] to efficiently find similar chemical fingerprints in a database based on Bloom filters and a user defined similarity threshold value. According to the experiments conducted by the authors, it was noted that the performance of the queries increased with the use of this tree data structure. The tree reduced the amount of comparison calculations and computationally scaled linearly when the size of the datasets was increasing.

  The concept of multi-bit trees was further examined by Bachteler et al. [12] as a new blocking method for record linkage. The string values in the attributes are first converted into sets of q-grams, which are then mapped into a Bloom filter. The generated Bloom filters are then partitioned into separate bins according to the number of bits set to 1 in them. All the Bloom filters in each bin are stored in a multi-bit data structure. A given Bloom filter is queried against all Bloom filters that are stored in the multi-bit tree and at each node the similarity is calculated to find matches.

  Similar to the approach by Bachteler et al. [12], Schnell suggested that records in a database can also be mapped into Bloom filters in PPRL. However, to increase the security of the Bloom filter encodings, rather than creating a Bloom filter for each identifier separately, Schnell suggested to include encodings of the set of attributes into one Bloom filter for each record. The resulting Bloom filter is called a *Cryptographic Long-term Key* (CLK) [189]. The generated CLKs are then stored in a multi-bit data structure which can be queried against to generate candidate record pairs. According to the experiments, Schnell showed that the proposed approach is having a linear computational complexity with increasing number of records in a dataset. Recently, several PPRL approaches [27, 188, 192, 194] have been suggested that use multi-bit tree based blocking.

- Following the same concept of using public reference values by Karakasidis and Verykios [111], Vatsalan et al. [215] suggested an efficient two-party private blocking technique based on a sorted nearest neighbourhood clustering approach which provides k-anonymous privacy. In this approach each DO needs to select a set of reference values from a publicly available database. The selected reference values are used to cluster the records in each database. Once the clusters are generated each DO merges them in order to ensure each cluster contains at least k records. This provides k-anonymous privacy characteristics, as each record in the database can be seen to be similar to at least k-1 other records.

  Once the k-anonymous clusters are generated, reference values corresponding to each cluster need to be exchanged between the DOs. By using the exchanged reference values the candidate clusters can be identified by each DO. The sorted nearest neighbourhood approach [95] is used to achieve this goal. Each of the candidate clusters that are identified will be considered as a separate block for the matching and classification steps. In their approach, linkage quality and privacy mainly depend on the reference values selected for blocking.

- A private blocking mechanism based on hierarchical clustering and differential privacy was introduced by Kuzu et al. [134] for two database protocols. Initially global clusters are generated for a set of reference values using hierarchical clustering. Then each DO assigns their records to these global clusters based on their similarity. Differential privacy is used to ensure privacy against inference due to clusters released between the DOs. Since the resulting clusters contain fake records that are added as noise, only a SMC-based private comparison and classification technique can be applied on the candidate record pairs. The approach is not scalable due to computationally expensive similarity calculations.

3.1.3 Multidatabase Protocols

As we have reviewed above, there have been various blocking approaches proposed for PPRL. However, most work so far has considered only on blocking records from two databases. Some approaches [91, 113, 115, 119] initially proposed for two database scenarios have also been investigated for blocking of three or more databases, although these techniques were never properly evaluated in terms of scalability to multiple databases.

- Recently, Han et al. [91] extended their two database dynamic k-anonymous blocking approach [90] to multiple databases using the Pallier cryptosystem. Similar to [90], the suggested blocking approach can only be applied on numerical attributes where numerical values are used as reference values to generate blocks. Each DO encrypts the reference values of its blocks by using a public key sent by a LU. Then all DOs participate in a ring communication to perform a homomorphic addition of the encrypted reference values for each block. Once the homomorphic addition is finished the first DO sent the summed encrypted reference values of each set of blocks to the LU to decide whether to compare records among multiple blocks based on their similarity. However, the proposed approach requires expensive computation and communication steps in measuring the similarity between blocks which makes the approach not scalable to multiple databases and large database sizes.

However, as we have discussed in Chapter 1, linking records from multiple databases is increasingly being required for Big Data applications, therefore creating an increased demand for scalable blocking techniques for MD-PPRL.

3.1.4 Filtering and Meta-blocking Techniques

Apart from these blocking techniques, filtering techniques could also be used to reduce the record comparison space by removing those record pairs that have a similarity below a given threshold value. In these filtering based PPRL approaches it is assumed that all DOs send their masked records to a LU to apply the filtering technique. For PPRL several such filtering based techniques [195, 196] have been proposed.

- In 2015 Sehili et al. [195] proposed a filtering technique based on the PPJoin technique [230], known as *P4Join*, for PPRL. In this approach the records in two databases are encoded into Bloom filters which are then compared for similarity. PPJoin is a signature-based similarity join algorithm that applies several optimisation techniques for improved efficiency, in particular a length filter, a prefix filter, and a position filter. The length filter remove those Bloom filter pairs that differ by a certain number of bits based on the predefined similarity threshold value. The prefix filter is based on the fact that similar Bloom filters need a high degree of overlap in their 1-bit positions in order to satisfy the similarity threshold. Pairs of records can thus be excluded from comparison if they have an insufficient overlap. The position filter can remove the comparison of two Bloom filters even if their prefixes overlap depending on the positions where the overlap occurs.

  The authors also investigated the applicability of parallisation of P4Join in a PPRL approach. P4Join based similarity calculations are performed in a graphical processing unit (GPU) where many calculations can be executed in parallel depending on the number of execution units in a GPU. The authors showed that the utilisation of GPUs can speed-up the similarity computations. Modern GPUs provide thousands of cores that allow for a massively-parallel application of the same instruction set to disjoint data partitions. The authors used an OpenCL[2] framework to simplify the utilisation of GPUs to parallelise algorithms.

[2]https://www.khronos.org/opencl/

- Recently, Sehili and Rahm [196] proposed a metric space based filtering approach which showed improved efficiency and linkage quality compared to previous PPRL approaches [190, 195]. In this approach records in two databases are encoded into Bloom filters (BFs). The BFs of the first database are added to the metric space by using a pivot based technique that selects a certain number of BFs from a sample of this database as pivots. Next, BFs are assigned to the closest pivot by computing the Hamming distance between the pivots and the BFs. Pivots are iteratively determined from the sample set of BFs such that the BF with the greatest distance to all previously determined pivots becomes the next pivot. Next, the BF of the second database is queried in the metric space by searching for the pivots within a given minimum distance. For the BFs of the relevant pivots the triangle inequality [216] is further used to prune BFs from the comparison with the query BF.

In addition, several block processing techniques have been proposed in record linkage (RL) and PPRL to reduce the record comparison space. These techniques are utilised in between the blocking and comparison step of the linkage process.

- Whang et al. [225] were the first to introduce an interactive block processing technique to reduce record comparison space for database de-duplication. In their approach, each individual block is repeatedly examined to detect duplicates. The result of record pair comparisons from each processed block is propagated to inform decisions in subsequent blocks. Two records in a block are merged if they are identified as a match and the resulting new merged record is propagated to other blocks. This approach can reduce the number of pair-wise comparisons in subsequent blocks since repeated comparisons are avoided. The same concept was extended further by Kim and Lee [124] who used locality sensitive hashing (LSH) for cleaning and merging datasets.

- Papadakis et al. [163] suggested a de-duplication approach using two categories of block processing methods to improve the performance of the linkage of two databases. Block refining methods, such as block purging (discard non-matching record pair comparisons by removing blocks that contain records more than a given upper size limit) and block scheduling (sort the set of blocks for comparison to reduce the number of duplicate record pair comparison), operate at the coarse level of processing individual blocks.

  The second category of comparison-refinement methods, such as comparison propagation (order the record pair comparisons to avoid any repetitive comparisons), duplicate propagation (remove record pairs in blocks if their are already been identified as duplicates), comparison pruning (remove any record pair comparisons that are unlikely to be match), and comparison scheduling (order the record pair comparisons of a given block such that record pairs that are likely to be matched are compared first), consider a finer level of block processing at the level of individual comparisons within blocks. The results presented by the authors showed that such refinement methods improve the efficiency of the overall de-duplication process.

- In 2014 Papadakis et al. [164] introduced the concept of *meta-blocking* for record linkage (RL). It aims to restructure a collection of blocks to reduce the number of record pairs within blocks. The suggested approach takes a block collection as input and uses a supervised classification technique based on block-feature vectors to distinguish promising comparisons. However, the suggested approach requires the selection of suitable features and training data to achieve accurate pruning of candidate comparisons. To improve the efficiency of meta-blocking further, Efthymiou et al. [64] have proposed a parallelised variation of the meta-blocking approach based on MapReduce [51]. Meta-blocking has recently been extended by Karakasidis et al. [109] for PPRL. In this approach, sorted neighbourhood blocking [95] based on reference values is used as a meta-blocking technique to further reduce the comparison space.

3.2 Classification Techniques in PPRL

The candidate record pairs or tuples that are generated in the blocking step are compared and classified into matches and non-matches in the next step of the PPRL process. The comparison and classification step addresses the linkage quality challenge by using private approximate comparison and classification techniques, while preserving the privacy of all non matching records in the databases that are being linked. In the following subsections we review the existing classification techniques under different database linkage scenarios.

3.2.1 Two Database Protocols with a Linkage Unit

- Van Eycken et al. [207] proposed a secure three-party protocol that uses an exact matching technique to classify health records from different databases. The suggested approach is based on exact matching of hash codes, which are generated by a secure one-way hashing function and a public key encryption algorithm in order to prevent dictionary attacks.

  In this approach, both database owners merge the values of their linkage attributes into a single hash string. These hash strings are then sent to a linkage unit (LU) to classify the records using a deterministic classification technique [35]. Experiments conducted on health databases showed that the accuracy of the classification increases if the concatenated string includes the full date of birth value. While this approach is cost effective, it is inappropriate in real-world applications since it can only perform exact matching of attribute values.

- Churches and Christen [42] proposed an approximate classification technique for performing secure record linkage on two databases based on hash encoding, public key encryption, and q-gram similarity comparison techniques. Using this approach PPRL can be performed *blindly* where neither of the DOs will need to share any information, even the meta-data of the databases, such as database schemas. A LU is used to conduct the linkage. To perform the attribute based comparison, the suggested approach uses subsets of q-gram sets of attribute values to calculate the Dice similarity between attribute values. These subsets are encoded and sent to the LU to classify each record pair as a match or non-match using a threshold based classification technique. However, the construction of the subsets of the q-grams is not feasible for real-world applications because of high computational requirements which makes the suggested approach not scalable to large databases.

- A protocol to provide privacy for both record and schema matching without revealing any information was presented by Scannapieco et al. [186]. This approach uses a set of reference values to embed records into a metric space while preserving the distances between attribute values of these records. An approximate comparison function is used to measure the distances between records. The computed distances are then sent to a LU to perform the linkage. The authors use a greedy heuristic re-sampling method for arranging records into blocks to improve the efficiency of the approach. However, the experimental results shows that the linkage quality is affected by this greedy heuristic re-sampling method. Recently, Bonomi et al. [19] extended this approach by using frequent q-grams of attribute values where the sample for the embedding is mined by the DOs using a prefix tree to satisfy differential privacy.

- A reference string based scalable approximate matching protocol was proposed by Pang et al. [162] based on an embedding approach. The authors use a common set of reference strings to find the matching record pairs. The DOs compute the distance between the reference strings and their values and send the results to a LU which can calculate the similarity between values using the triangular inequality property. The performance of this approach depends on the set of reference strings used.

- Weber et al. [223] have proposed a simple heuristic method for privately linking medical data using a LU. They hypothesised that the use of encoded identifiers based on name prefix and date of birth provides better comparison results than using identifiers with full name and social security number (SSN). String values of the identifiers are encoded using a secret key shared between the DOs, and the encoded values are sent to a LU to perform the comparison. According to the experimental results, the authors concluded that composite identifiers perform better in terms of sensitivity and specificity than using identifiers with full name or SSN. The composite identifiers are useful for situations where the exchange of these identifiers is impossible.

3.2.2 Two Database Protocols without a Linkage Unit

- Atallah et al. [9] proposed an approximate matching protocol for securely comparing sequences in genome sequence databases. The suggested approach uses the edit-distance [35] method to calculate the similarity between sequences. The edit-distance matrix is split between the DOs such that the minimum cost of the edit-distance is calculated additively by each DO. A blind and permute protocol, which uses a homomorphic encryption scheme, is introduced to minimise the information learned by each DO. However, this approach is not appropriate for linkage of large databases due to excessive communication overhead required to compute the edit-distance matrix.

- Freedman et al. [76] proposed an efficient privacy-preserving keyword search algorithm for Privacy-Preserving Information Retrieval (PPIR) applications. The proposed approach considers the linkage of two databases under both the honest-but-curious (HBC) and malicious adversary models. For a given keyword by a client, the proposed approach uses homomorphic encryption and oblivious pseudo-random functions to search for matching keyword and payload pairs in a database. The theoretical proofs provided by the authors show that the proposed approach is having a linear computational complexity and a poly-logarithmic communication complexity in the size of the databases.

- Yakout et al. [231] proposed an approximate classification technique based on the work by Scannapieco et al. [186]. Similar to [186], this approach transforms records into objects in an embedding metric space using a set of reference values, while preserving the distances between records. In this approach attribute values are represented as vectors. Using these vectors, complex numbers are calculated to create a complex plane and the likely matched pairs are computed by moving an adjustable width slab across this complex plane. The Euclidean distance [35] is used to measure the approximate similarity between records. Based on these distances, similar record pairs are classified as those that are within the slab width. These similar pairs are then compared in detail in the second phase of the protocol using a secure scalar product protocol based on randomised vectors. Experimental results showed that the scalability and linkage quality depend on the slab width.

- Li et al. [137] proposed a classification technique for privacy-preserving group linkage (PPGL) to measure the similarity of groups of records rather than individuals. Threshold-based PPGL methods were proposed for both exact and approximate matching. The proposed approach is adopted for both the HBC and malicious adversary models. The authors developed two protocols for exact matching with commutative and homomorphic encryption, which use Jaccard similarity to calculate the group level similarity. The approximate matching protocol uses the Cosine similarity [229] measure with a private scalar product protocol to find the similarity between groups. According to the experiments results the proposed approach does not scale with an increase of group size due to its exponential computational complexity.

- Vatsalan et al. [214] proposed an approximate classification protocol which follows the HBC adversary model. The authors used a set of reference values, which are extracted from a publicly available database agreed upon by both DOs, for calculating the similarities between attribute values. To improve the scalability a phonetic based blocking technique is used. Each DO calculates similarities between the reference values and the linkage attribute values which are then generalised into bins. The bin values are securely exchanged between the two DOs. Record pairs that have the same similarity bin combinations in their attributes, or that have a similarity binning distance less than a maximum binning distance, are classified as matches. However, the number of bins and bin value ranges need to be chosen carefully, as they provide a trade-off between scalability, privacy, and linkage accuracy.
- An iterative approximate classification protocol was recently proposed by Vatsalan and Christen [208] which reveals selected bits in records encoded into BFs between two DOs. The approach classifies record pairs into matches, non-matches, and possible matches in an iterative way to reduce the number of record pairs with unknown match status at each iteration without compromising privacy. At each iteration the minimum similarity based on the revealed bit positions is calculated using the Dice-coefficient. The authors used a length filtering technique to reduce the number of record comparisons.

3.2.3 Multidatabase Protocols

- O'Keefe et al. [160] proposed a secure multi-party computation (SMC) based exact classification approach for multidatabase PPRL and privacy-preserving extraction of a cohort of an individual's data from a database without revealing the identity of these individuals to the DOs. This approach improves on the security and information leakage characteristics compared to an earlier approach developed by Agrawal et al. [2] which provided the basis for the proposed approach. The proposed approach is however computationally expensive due the use of SMC based calculations. Though the initial approach was proposed for linking of two databases, the authors have also proposed an extended version of the protocol, where more than two database owners can link their data privately. However, the proposed approach is not capable of linking records that contain errors and variations in the attribute values.
- A multidatabase classification approach was proposed by Lai et al. [135] that uses Bloom filters (BFs) to securely compare records between multiple databases for private set intersection. All records in a database are encoded into one Bloom filter (BF) and segmented according to the number of DOs involved in the protocol. These segments are shared among the parties, where each DO receives segments of the BFs from all the other DOs and then performs a logical

conjunction (AND) on its received segments. These logically conjuncted segments are shared across all the DOs to combine into a one conjuncted BF. Each DO compares its own full BF with the conjuncted BF for identifying matches. Evaluation results showed that the false positive rate increases with the number of DOs involved in the protocol, where none of the DOs will be able to guess the existence of a given record correctly.

- Kantarcioglu et al. [108] proposed a multidatabase approach based on the k-anonymity [216] generalisation technique for person specific biomedical data. This approach performs efficient secure joins of encrypted databases by a linkage unit via encrypted identifiers. The DOs perform a generalisation algorithm using k-anonymity based on the common set of quasi-identifying attributes. Before the linkage unit performs the equi-join operation, blocks are constructed corresponding to each combination of k-anonymous values. The number of secure equi-joins required by the protocol is drastically reduced when a k-anonymous equi-join is applied, compared to the full comparison of all record sets. However, the suggested approach has a complexity of $O(n^2)$ for applying generalisation to each database with n records and required significant communication between the database owners and the linkage unit which make this approach not scalable to multiple databases. Nevertheless the proposed approach is only applicable to categorical data, though the proposed approach can also be used for other data types by using proposer discretisation techniques [89].

- Mohammed et al. [152] proposed an approximate classification approach for multidatabase PPRL using the k-anonymity based generalisation privacy technique without the need of a linkage unit. This work is based on the secure distributed k-anonymity framework proposed by Jiang and Clifton [104] for integrating two private databases into a k-anonymous database. The proposed approach uses game theory concepts to prevent DOs from sending false values in a malicious adversary model. The proposed approach uses an adapted version of the C4.5 decision tree classifier to recursively block and classify the records in databases. According to the experimental results the proposed approach outperforms [104] in terms of algorithmic complexity and scalable for large database sizes. However, the proposed approach does not scale for multiple number of databases because of the expensive communication required by the participating database owners.

- Vatsalan and Christen [210] proposed a multidatabase approximate classification approach following the work by Lai et al. [135]. The protocol combines the BFs, secure summation, and the Dice coefficient with the aim to identify all records held by the different DOs that have a similarity above a certain threshold. The protocol has a communication complexity that is linear in the number of DOs and the size of the databases that are linked, however, it requires an exponential number of record comparisons which makes the protocol not scalable to applications where records from many databases need to be linked.

- In 2015 Karapiperis et al. [114] proposed a multidatabase PPRL approach for categorical data using a Count-Min sketch data structure (as was defined on page 28). Sketches are used to summarise the local set of elements which are then intersected to provide a global synopsis. First each DO independently summarises its records in a local synopsis, which is implemented by a Count-Min sketch. Then these local synopses are intersected in order to create the global synopsis. This global synopsis provides collective count estimates for the common elements attained in those databases and hides the contribution of each DO to these estimates. The authors proposed two protocols for generating global synopsis where the first one uses homomorphic operations that provide improved privacy and accuracy with high communication costs, and the second protocol uses a secure summation protocol which exhibits improved efficiency.

- In 2016 Vatsalan et al. [213] proposed a multidatabase protocol for PPRL that is based on Counting Bloom filters (CBFs). The proposed approach allows approximate matching of attribute values that are encoded into BFs. To enable more secure approximate matching CBFs are used to count the number of 1's in each bit position of a given set of BFs. To sum the number of 1's in each bit position this approach uses a secure summation protocol which is initiated by a linkage unit (LU). A secure summation protocol is performed for each record in a database which makes the protocol communicationally expensive.

  Based on the CBF of each set of BFs the LU computes the Dice similarity to classify the corresponding set of records as either matches or non-matches based on a similarity threshold value. To improve the scalability of the approach the authors used a Soundex based phonetic blocking technique for each database. To reduce communication overhead and the number of required computations the authors proposed a ring based communication pattern where DOs are grouped into rings. The experiments using real datasets showed that the proposed approach is scalable in terms of database size and provides improved privacy compared to earlier approaches [135, 210].

- Recently, Saeedi et al. [185] conducted a comparative evaluation of clustering schemes for multidatabase record linkage. The authors evaluated different clustering techniques, including correlation [32], center [93], merge center [93], and star [8] clustering, upon a framework called *fast multi-source entity resolution system* (FAMER). This system accepts a set of databases as input and outputs a collection of clusters. All records within a cluster is considered as matches (belongs to the same entity) while different clusters refer to different entities.

  In FAMER, first the records are blocked into a set of groups and a similarity graph is computed based on the comparison between records in these blocks. A clustering technique is then applied upon the similarity graph to group entities such that the similarity between entities within a cluster is maximised while the similarity between entities of different clusters is minimised. The authors also proposed parallelised versions of the clustering techniques using Apache

Flink [3]. The experiments using real and synthetic datasets showed that the use of clustering improves the linkage quality while the scalability of linkage can be improved by parallelising the clustering techniques.

3.3 Summary

In this chapter we have presented a survey of historical and current state-of-the-art techniques for PPRL. As detailed in this chapter, most of the current blocking techniques proposed for PPRL can only block two databases. Nevertheless, some classification protocols proposed for multiple databases mainly depend on traditional blocking techniques, which are not secure as we shown in Chapter 10, while others do not use any blocking technique to scale to multiple databases. Therefore, the development of scalable and secure blocking techniques that can block multiple databases (more than two) is an important requirement in any multidatabase linkage that has not been addressed in enough details so far. In the next chapter we will propose a novel blocking framework for multidatabase PPRL to address the challenges related to multidatabase linkage, as described in Chapter 1.

[3]https://flink.apache.org/

Chapter 4

A Blocking Framework for Multidatabase Privacy-Preserving Record Linkage

As we discussed in Chapter 1, the non-availability of a blocking framework is a major limitation we have identified in a multidatabase linkage context. Having such a framework would be highly beneficial for RL and PPRL practitioners to identify different methods and techniques that can be used to block multiple databases under various real-world scenarios. In this chapter we address this limitation by introducing a blocking framework that can be used in multidatabase privacy-preserving record linkage (MD-PPRL).

In Section 4.1 we highlight the importance of a framework for blocking in a MD-PPRL context and in Section 4.2 we describe our framework. In Section 4.3 we then provide details on different building blocks that are used for developing the techniques used in our framework. In Section 4.4, we provide details about different datasets and measures that are used for evaluation in our framework, and the computational platform that will be used to evaluate our proposed framework. Section 4.5 provides a detailed description about an attack method that will be used for privacy analysis of our proposed techniques. In Section 4.6, we provide an outline of our framework. Table 4.1 summarises the notation we use in this chapter.

4.1 Introduction

As discussed in Chapter 1, blocking is an important step in the PPRL process [35, 216]. Blocking makes the linkage process scalable by grouping similar records into blocks while ensuring dissimilar records are inserted into different blocks. As we have reviewed in Chapter 3, over the past two decades many blocking techniques have been introduced for PPRL. These blocking techniques use different grouping methods such as clustering [111, 162], reference values [109, 111], hashing [118, 119, 127], or embedding [108]. Below, we consider the usability and applicability of these existing blocking techniques in a multidatabase PPRL context.

Table 4.1: Notation and terminology used in this chapter

$\mathbf{B}$	A set of blocks
$\mathbf{D}$	A database
$\mathcal{E}$	Total summed value
$\mathbb{G}$	Global (publicly available) database
B	A block of records
CBT	Candidate block tuple
CRT	Candidate record tuple
DR	Disclosure risk value
$\mathcal{H}$	Set of hash functions
PS	Probability of suspicion value
Q	A queue data structure
$R, R.a, R.id$	A record, record attribute value, and record identifier
S	A set of q-grams
ε	An encrypted value
d	Number of database owners (or databases)
fp	False positive rate
$h(\cdot)$	A hash function used for mapping a value into a Bloom Filter
n_h, n_q, n_r	Number of hash functions, q-grams, and records
l_q, l_{bf}	Length of a q-gram and a Bloom filter
q	A q-gram
r	A random number
sp	A partially summed value
BF	Bloom filter
BKV	Blocking key value
DO	Database owner
LU	Linkage unit
MD-PPRL	Multidatabase PPRL
MDRL	Multidatabase record linkage
PC	Pairs completeness
PQ	Pairs quality
PPRL	Privacy-preserving record linkage
RR	Reduction ratio
SMC	Secure multi-party computation

First of all, most of the blocking techniques that have been proposed for PPRL are capable of blocking only two databases [56, 111, 215]. These blocking techniques are not capable of blocking multiple databases which creates a necessity in developing blocking techniques for a MD-PPRL context. Also, these existing blocking techniques do not provide flexibility and control for the database owners (DOs) in the block generation process on their own databases since all the DOs need to agree on the same parameter settings, such as the number of blocks to be generated and their sizes, when blocking their databases [36, 165].

Secondly, subgroup blocking is an important aspect that needs to be considered for a multidatabase record linkage (MDRL). As we explained in Chapter 1, an analysis of subgroup populations can be important as they describe different features and aspects within a larger population [22, 131]. In a MDRL context subgroups of databases need to be linked to identify the subsets of matching records in different databases. The existing blocking techniques that have been proposed for PPRL are not capable of performing blocking for subgroups of databases.

Thirdly, in a PPRL context, these existing blocking techniques result in a record comparison space where most of the record pair comparisons are potentially identified as non-matches [164]. These record pair comparisons incur additional computational costs in linkage applications which creates a need for efficient filtering techniques that can be incorporated with blocking to remove redundant record pair comparisons. In general, such filtering techniques can be employed in between the blocking and comparison steps of the linkage process [163]. Several filtering techniques have been proposed for the de-duplication of a database, however, these techniques are not generally suitable for MD-PPRL since they do not guarantee the privacy of records that are linked (we will provide more details about these existing filtering techniques in Chapter 9).

Finally, a framework that can be used to combine these three different aspects of (1) control and flexibility in block generation, (2) subgroup blocking, and (3) filtering of redundant record comparisons in blocking has not yet been developed for MD-PPRL. Each of these aspects has its own objective, however, they are interrelated in the context of blocking. A combination of these three aspects into a framework will generate a flexible blocking platform for various PPRL applications where different techniques could be used to maximise the overall efficiency and effectiveness in linking multiple databases.

As we have reviewed in Chapter 3, over the past two decades several frameworks have been proposed in PPRL. Durham proposed a linkage framework for performing private record linkage between two databases [56]. The proposed framework employs a three-party linkage scenario (as was described in Chapter 2) where the encoded records of the two databases are sent to a trusted third party (such as a linkage unit, LU) to identify the matching records. However, the proposed framework is only applicable in the context of linking two databases and does not provide any functionalities for subgroup blocking or efficient filtering of unwanted record comparisons.

Papadakis et al. [163] recently proposed a blocking framework for record linkage on two databases. The proposed framework considers blocking of databases with heterogeneous schemes, and provides a filtering technique that can be used to remove record comparisons that can be considered as non-matches. However, blocking of multiple databases and identification of blocks that need to be compared in subgroups of databases are not supported in this proposed framework.

Recently, Karapiperis and Verykios proposed a blocking framework for PPRL of two databases [117]. The proposed framework uses *locality sensitive hashing* (LSH) [101] for grouping similar records into blocks. Each block is then sent to a LU to compare

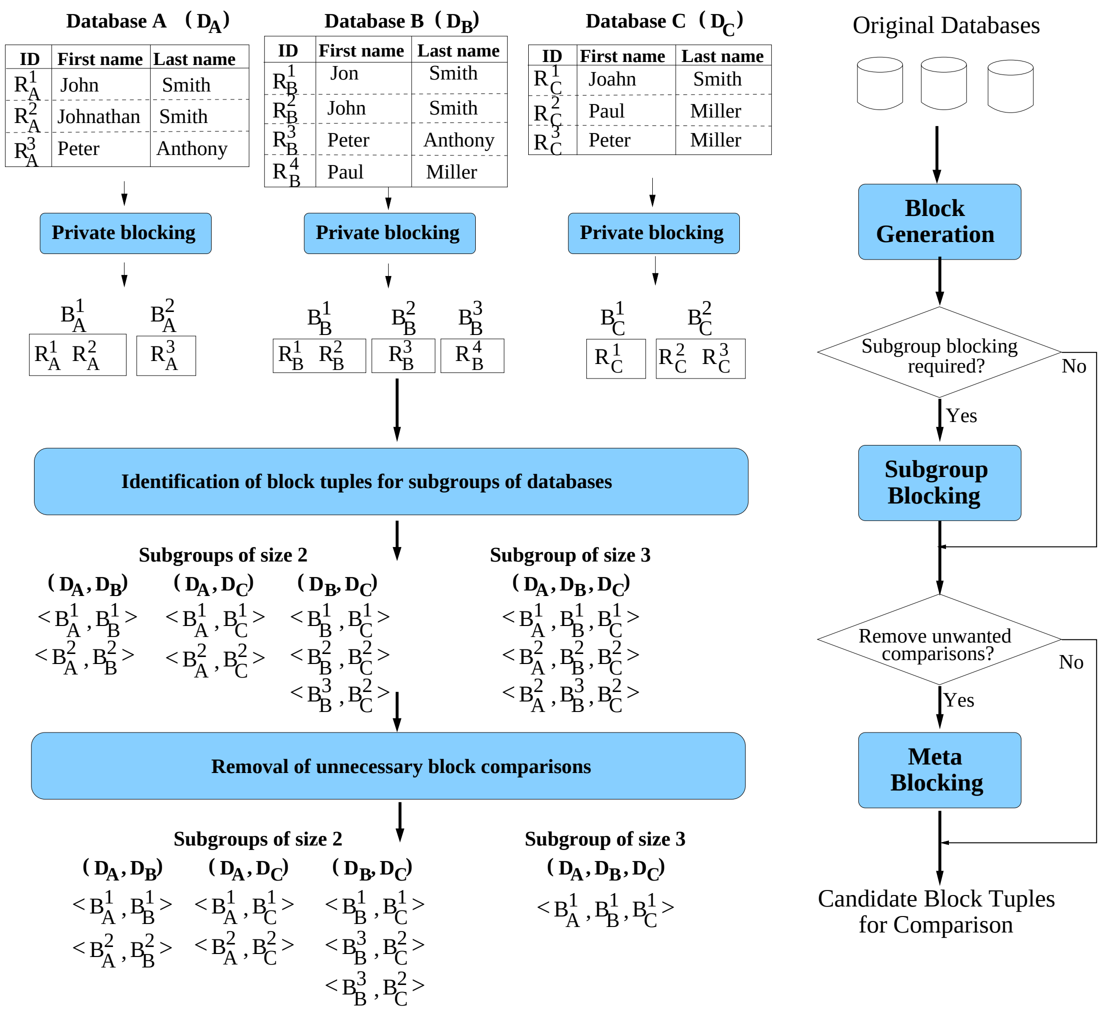

Database A (D_A)
Database B (D_B)
Database C (D_C)
ID | First name | Last name
Private blocking
Identification of block tuples for subgroups of databases
Subgroups of size 2
Subgroup of size 3
Removal of unnecessary block comparisons
Original Databases
Block Generation
Subgroup blocking required?
Yes
No
Subgroup Blocking
Remove unwanted comparisons?
Meta Blocking
Candidate Block Tuples for Comparison

Figure 4.1: An overview of our multidatabase blocking framework. The left side represents an example of blocking of three databases of $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}_C$, while the right side represents each corresponding layer with its functionality in our framework. We use the notation B_A, B_B, and B_C to represent blocks of $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}_C$, respectively. In the block generation layer, the records are assigned into blocks by using an appropriate blocking technique. These generated blocks can then be input to subgroup blocking layer where similar blocks that need to be compared for different subgroup combination of these databases are identified. Finally, these identified blocks can be input to the third layer, called meta-blocking, to remove unnecessary block comparisons that reduce the number of unwanted record comparisons.

record pairs from the same blocks to identify matches. The proposed framework is built on top of a Map/Reduce paradigm in order to distribute computations among underutilised commodity hardware resources uniformly. However, the proposed framework does not support linking of multiple databases, subgroup blocking, or removal of unwanted record pair comparisons.

4.2 Framework Overview

To facilitate the three different aspects of blocking mentioned in Section 4.1, we propose a blocking framework that can be used for MD-PPRL. The main aim of our framework is to provide a flexible platform for PPRL researchers and practitioners to work with different techniques that can be used for blocking of multiple databases. As a goal this framework allows users to employ different blocking techniques that can be used appropriately under different MD-PPRL scenarios. We next describe our framework in more details.

As illustrated in Figure 4.1, we consider three main aspects of multidatabase blocking, which are (1) the generation of blocks of each database to be linked, (2) identifying blocks that need to be compared across two or more databases, and (3) reducing the number of record pair comparisons by removing redundant and superfluous comparisons as we will discuss in Chapter 9, as separate functionalities. To facilitate each functionality, we follow a layered approach where each layer consists of several techniques that support a specific functionality. Each layer receives as input the output of the previous layer with the aim to produce an output that improves either effectiveness or efficiency (or both aspects) of the input. Since each layer comprises multiple techniques, the techniques provided in each layer can be combined to perform different multidatabase blocking workflows. Hence, users are provided with flexibility to select different combinations of techniques suitable for their linkage scenario to maximise scalability, blocking quality, and privacy.

Our framework accepts d databases as input, and outputs a set of *candidate block tuples* (**CBT**) that need to be compared using a private comparison and classification technique (as described in Chapter 2). Each candidate block tuple (CBT) contains blocks from at least 2 databases to at most d databases with a maximum of one block from each database. We define a CBT as:

Definition 4.1. Candidate block tuple (CBT)
Assume $\mathbf{B}_1, \mathbf{B}_2 \cdots, \mathbf{B}_d$ are sets of blocks from respective databases $\mathbf{D}_1, \cdots, \mathbf{D}_d$ held by database owners DO$_1, \cdots,$ DO$_d$. A candidate block tuple $CBT = \langle B_i^x, \cdots, B_j^y \rangle$, where $B_i^x \in \mathbf{B}_i, B_j^y \in \mathbf{B}_j$, $1 \leq i, j \leq d$, $i \neq j$, $1 \leq x \leq |\mathbf{B}_i|$, $1 \leq y \leq |\mathbf{B}_j|$ and $2 \leq |CBT| \leq d$.

The comparison of records in each block in a given CBT of size $2 \leq |CBT| \leq d$ results in a set of candidate record tuples where each candidate record tuple (CRT) can be defined as:

Definition 4.2. Candidate record tuple (CRT)
Assume $\langle B_i^x, \cdots, B_j^y \rangle$ is a candidate block tuple CBT, with $2 \leq |CBT| \leq d$, that needs to be compared from the respective databases $\mathbf{D}_1$, $\mathbf{D}_2$, $\cdots$, $\mathbf{D}_d$. A candidate record tuple $CRT = \langle R_i^m, \cdots, R_j^n \rangle$, where $R_i^m \in B_i^x$, $R_j^n \in B_j^y$, $1 \leq m \leq |B_i^x|$ and $1 \leq n \leq |B_j^y|$.

It is important to note that our framework provides the capability for each layer to function independently. The framework does not require all three layers to be combined together to generate the blocks for comparison and classification, but once coupled they increase either the effectiveness, the efficiency, or both of the overall linkage. Depending upon the availability of computational resources or cost budgets these layers can be de-coupled as shown in Figure 4.1.

As can be seen in the right-hand side of Figure 4.1, a user is given the opportunity to decide the next functionality they require once the blocks are formed for their databases. These generated blocks can be input to either the second or third layers, or can be used directly for comparison. Following the same principle the output of the second layer can be used for comparison without using any functionality from the third layer. We next describe the layers of our proposed framework in more detail.

4.2.1 Layer 1: Block Generation

As illustrated in Figure 4.1 the first layer, named *block generation*, is aimed at blocking the records in each database to be linked into a set of blocks. The block generation layer accepts databases as input and outputs a set of blocks $\mathbf{B}_i$ for each database $\mathbf{D}_i$. The block generation layer can be divided into three different steps.

1. *Parameter setup*: Based on the technique used in the block generation layer, certain parameters likely need to be set. As shown in Figure 1.3, in MD-PPRL without a LU all the DOs need to agree upon the set of parameters to be used. On the other hand the parameters can be set by the LU if the blocking of these databases is conduced by a LU.

2. *Database masking*: Once the parameters are agreed, each record of these databases needs to be masked into an appropriate encoding format in accordance with the technique to be used in the block generation process. For masking of individual records different encoding techniques can be used as described in Chapter 2.

3. *Blocks construction*: The encoded records in those databases are grouped into blocks by using an appropriate blocking technique available in the framework. We propose three different multidatabase blocking techniques that can be used in this layer and they are described in Chapters 5 to 7 in more detail.

As can be seen in the left-hand side of Figure 4.1, once the block generation process is finished, blocks $\mathbf{B}_A = \{B_A^1, B_A^2\}$, $\mathbf{B}_B = \{B_B^1, B_B^2, B_B^3\}$, and $\mathbf{B}_C = \{B_C^1, B_C^2\}$ have been generated from the databases $\mathbf{D}_A, \mathbf{D}_B$, and $\mathbf{D}_C$, respectively.

4.2.2 Layer 2: Subgroup Blocking

As shown in the right-hand side of Figure 4.1, the block generation layer outputs sets of blocks for each database as input to the second layer which we name *subgroup blocking*. The aim of this second layer is to generate the CBTs that need to be compared across different subgroups of the databases to be linked. Assuming d databases are to be blocked, our framework is capable of generating CBTs for $\sum_{i=2}^{d} \binom{d}{i}$ subgroup combinations. The subgroup blocking layer outputs a list of sets where each set contains CBTs that need to be compared for a given subgroup combination of the databases that need to be linked. Our subgroup blocking techniques are described in more detail in Chapter 8.

As can be seen in Figure 4.1, for databases $\mathbf{D}_A, \mathbf{D}_B$, and $\mathbf{D}_C$ the subgroup blocking layer generates CBTs for subgroup combinations $(\mathbf{D}_A, \mathbf{D}_B)$, $(\mathbf{D}_A, \mathbf{D}_C)$, and $(\mathbf{D}_B, \mathbf{D}_C)$ and $(\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C)$, of sizes 2 and 3, respectively. The user can then input these generated CBTs to the third layer, which we will be described next.

4.2.3 Layer 3: Meta-blocking

As we discussed in Chapter 1, MDRL requires an exponential number of record pair comparisons even after a blocking technique has been applied. The third layer of our framework, called *meta-blocking*, focuses on addressing this problem by reducing the number of redundant record pair comparisons in a given set of CBTs to improve the efficiency of a linkage with no loss in effectiveness.

We consider two categories of redundant record comparisons, which are *repeated* and *superfluous* comparisons (as will be formally defined in Chapter 9), to be removed from the overall record comparison space. Repeated record comparisons occur when the same pair of records is compared repeatedly for multiple CBTs, while superfluous comparisons occur between records in a CBT where previously a non-match classification has been made, therefore rendering further comparisons with other records in the same CBT unnecessary as they cannot lead to a set of matching records across databases. We propose several techniques that can be used to remove these unwanted CRTs which will be described in more detail in Chapter 9.

As shown in the example in Figure 4.1, the block comparisons $\langle B^2_A, B^2_B, B^2_C \rangle$ and $\langle B^2_A, B^3_B, B^2_C \rangle$ of subgroup $(\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C)$ are removed from the block comparison space since they lead to unwanted record comparisons, for example, if there are no matching records found for block pair comparison $(B^2_A,\ B^2_C)$. This improves the efficiency and effectiveness of the overall linkage of the three databases $\mathbf{D}_A, \mathbf{D}_B$, and $\mathbf{D}_C$, without any loss in blocking quality.

4.3 Building Blocks used in Blocking Framework

Since each layer in our framework contains multiple techniques that can be employed in blocking, some building blocks (components and methods) used are common across those techniques. In this section we describe the building blocks used in our framework.

4.3.1 Q-grams

We use character q-grams (also known as n-grams) in our blocking approaches to encode attribute values (also known as blocking key values, BKV) to Bloom filters (which will be described next). A q-gram is a character sub-string of length l_q in a given string such as a first name or an address [35]. A q-gram of length 2 is

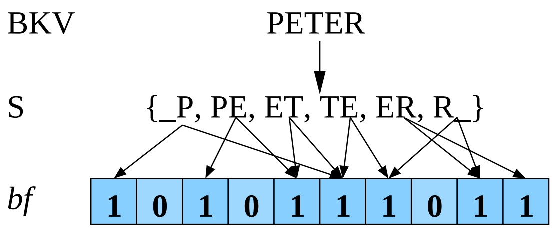


Figure 4.2: Mapping of a padded BKV value *PETER* into a Bloom filter of l_{bf} = 10 bits by using $n_h = 2$ hash functions.

known as a *bigram* or *digram*, while a q-gram of length 3 is known as a *trigram*. Often string values are prefixed and suffixed, which is also known as padding, with a special character before they are converted into q-grams. This padding character helps to identify the first and last characters of a string. A string of length l_s contains $(l_q - (l_q + 1))$ q-grams. For example, the string *PETER* can be padded with character '_' and the resulting *bigram* $(n_q = 2)$ set is $S = \{\_P, PE, ET, TE, ER, R\_\}$. We use q-grams in the masking step of the blocking techniques described in Chapters 5 to 7.

4.3.2 Bloom Filters

As detailed in Chapter 2, Bloom filters (BFs), which were proposed by Bloom [18], have widely been used for encoding of records in PPRL approaches [59, 190, 193, 208]. BF encoding is commonly used in PPRL for matching records due to its efficiency for encoding elements of a set and for querying if an element is a member of a set hashed into a BF [26].

A BF is a bit vector of length l_{bf}, where initially all the bits are set to 0. In order to hash (encode) an element into the domain between 0 and $l_{bf} - 1$ of a BF, n_h independent hash functions $\mathcal{H} = \{h_1, h_2, \cdots, h_{n_h}\}$ are used. Furthermore, to store n elements of the set $S = \{s_1, s_2, \cdots, s_n\}$ into a BF, each element $s_i \in S$, $1 \leq i \leq n$, is encoded using these n_h hash functions, and all bit (index) positions $h_j(s_i)$, $1 \leq j \leq n_h$, are set to 1. In our blocking approaches we encode the BKVs of each record into one BF. Figure 4.2 illustrates the encoding of a BKV into a BF. We use BFs in Chapters 5 to 7.

4.3.2.1 Optimal Parameter Settings for Bloom Filters

When generating BFs, it is crucial to use the optimal parameter settings in generating BFs which balances the aspects of blocking quality, scalability, and privacy in PPRL applications [189]. In this section we describe the calculation of optimal parameter settings for BF encodings based on the attribute values in a given database.

For a given BF length, l_{bf}, and the expected number of q-grams generated for an attribute value, n_q, the optimal number of hash functions, n_h, that minimises the false positive rate, fp, can be calculated as [150]

$$n_h = \frac{l_{bf}}{n_q} ln(2), \tag{4.1}$$

leading to a false positive rate of

$$fp = \left(\frac{1}{2^{ln(2)}}\right)^{\frac{l_{bf}}{n_q}}. \tag{4.2}$$

We can calculate the value for n_q by analysing the attribute values of blocking keys of a database that needs to be blocked in PPRL by calculating the average number of q-grams that are generated from a record (i.e. we convert attribute values into q-grams as described in Section 4.3.1 and count how many q-grams are generated on average for a record).

For a given BF length l_{bf}, we can calculate n_h based on n_q as calculated from the database. For a given database and n_q, longer BF (larger l_{bf}) will require more hash functions. However, a larger n_h would require more computations as more hash values need to be calculated and encoded into a BF for each attribute value in a record.

While n_h and l_{bf} determine the computational aspects of generating BFs, quality and privacy will be determined by the false positive rate fp. A higher fp value would result in a larger number of false positive matches (i.e. a set of non-matching records classified to correspond to the same entity), and thus lower quality of the linkage. At the same time, a higher fp will improve privacy, as it is more difficult for an adversary to identify a record that corresponds to a certain BF with a higher false positive rate [56, 190, 208].

It was shown by Mitzenmacher et al. that a BF should ideally have half of its bits set to 1 (i.e. 50% filled) to achieve the lowest possible false positive probability for given values of n_q, l_{bf} and n_h [150]. Equations 4.1 and 4.2 in fact lead to a probability that a bit position i of bf is set to 1 as [150]:

$$P_r[bf[i] \equiv 1] = e^{\frac{-n_h \cdot n_q}{l_{bf}}} = 0.5 \tag{4.3}$$

In PPRL, using optimal settings is important, because the bit patterns and their frequencies in a set of BFs can be exploited by a cryptanalysis attack [39, 134]. Such an attack exploits the fact that less frequent 1 bits in BFs can provide information about rare q-grams and thus rare attribute values. In our experimental evaluation we will set the BF parameters for our approach according to the discussion presented here and following earlier BF based approaches in PPRL [190].

Algorithm 4.1: Basic secure summation protocol (BSS)

```
Input  : $d$ - Number of DOs
         $n_i$ - The secret input of DO$_i$, $1 \leq i \leq d$
Output: $\mathcal{E}$ - Final sum where $\mathcal{E} \leftarrow \sum_1^d n_i$
DO$_1$ generates a random number $r$
DO$_1$ computes partial sum $sp_1 \leftarrow n_1 + r$
DO$_1$ sends $sp_1$ to DO$_2$
DO$_2$ receives $sp_1$
for $i \in [2, 3, \cdots, d]$ do                 // DO$_2$ to DO$_d$ follow the same steps
    DO$_i$ computes partial sum $sp_i \leftarrow sp_{i-1} + n_i$
    if $i == d$ then                              // Check if the current DO is DO$_d$
        DO$_i$ sends $sp_i$ to DO$_1$
        DO$_1$ receives $sp_i$
    end
    else
        DO$_i$ sends $sp_i$ to DO$_{i+1}$
        DO$_{i+1}$ receives $sp_i$
    end
end
DO$_1$ computes final sum $\mathcal{E} \leftarrow sp_d - r$   // Substract $r$ from final partial sum
DO$_1$ sends final sum $\mathcal{E}$ to other DOs
Other DOs receive final sum $\mathcal{E}$
```

4.3.3 Secure Summation protocols

As detailed in Section 2.4, secure summation is a secure multi-party computation (SMC) protocol that can be used to calculate a global summation over private numerical inputs held by multiple DOs [45]. We use the secure summation protocol as a communication protocol in our blocking approaches in Chapters 5 and 6.

As we detailed in Chapter 2, secure summation protocols are susceptible to collusion risk [45]. In the following sub-sections, we describe several existing secure summation protocols and we then propose a new protocol that exhibits improved security with regard to different types of collusion compared to these existing protocols. We provide an algorithmic description of each protocol, where we assume the two inputs to each protocol are: $d \geq 3$, the number of participating DOs, and the private numerical input to the protocol by each DO$_i$, n_i, with $1 \leq i \leq d$. Each protocol securely calculates and returns the final sum $\mathcal{E} = \sum_{i=1}^d n_i$ without revealing any of the n_i to any other DOs.

4.3.3.1 Basic Secure Summation Protocol (BSS)

Algorithm 4.1 describes the steps involved in this basic protocol [45, 122]. Initially DO$_1$ chooses a large random number r (line 1) which it keeps secret from all other DOs. It then adds r to its input n_1 (line 2) to generate the partial sum sp_1. Then DO$_1$ sends sp_1 to DO$_2$ (line 3). Since r is random, DO$_2$ can not learn anything about n_1.

Algorithm 4.2: Encrypted secure summation protocol (ESS)

```
Input  : $d$ - Number of DOs
         $n_i$ - The secret input of DO$_i$, $1 \leq i \leq d$
Output: $\mathcal{E}$ - Final sum where $\mathcal{E} \leftarrow \sum_1^d n_i$
1  LU generates $pk$ and $sk$ of public and private key pair, respectively
2  LU sends $pk$ to all DOs
3  DOs receive $pk$
4  for $i \in [1, 2, \cdots, d]$ do                          // All DOs follow the same steps
5      $\varepsilon_i \leftarrow E(n_i, pk)$                  // Encrypt the secret input
6      if $i == 1$ then                                     // Check if the current DO is DO$_1$
7          DO$_1$ sends $\varepsilon_1$ to DO$_2$
8          DO$_2$ receives $\varepsilon_1$
9      end
10     else
11         DO$_i$ computes partial sum $sp_i \leftarrow sp_{i-1} + \varepsilon_i$
12         if $i == d$ then                                 // Check if the current DO is DO$_d$
13             DO$_d$ sends $sp_d$ to LU
14             LU receives $sp_d$
15         end
16         else
17             DO$_i$ sends $sp_i$ to DO$_{i+1}$
18             LU receives $sp_i$
19         end
20     end
21 end
22 LU gets the $\mathcal{E} \leftarrow D(sp_d, sk)$          // Decrypt the received patial sum value
23 LU sends final sum $\mathcal{E}$ to DOs
24 DOs receive final sum $\mathcal{E}$
```

DO$_2$ then adds its value n_2 to sp_1 $(n_1 + R)$, and sends the result to DO$_3$. This process is repeated (lines 5 to 15) until all the DOs have added their values, and the partial sum $sp_d = r + n_1 + \cdots + n_d$ is received by DO$_1$ (line 9). Now DO$_1$ subtracts r from sp_d (line 16) and the resulting sum $\mathcal{E}$ is distributed to all other DOs (line 17). Figure 2.2 on page 32 shows an example of BSS for three DOs.

4.3.3.2 Encrypted Secure Summation Protocol (ESS)

The idea of this protocol is to use a public and private key pair for encrypting and decrypting of private inputs, respectively, as detailed in Algorithm 4.2 [213, 232]. ESS uses the partially homomorphic Paillier cryptosystem [161]. As detailed in Chapter 2, homomorphic encryption is a reliable SMC technique for performing secure computation among several DOs.

In this protocol a linkage unit (LU), a trusted third party, generates the public (pk) and private (sk) key pairs. We assume a LU is employed to facilitate the secure summation across d DOs. The LU sends pk to all DOs to encrypt their private inputs, while sk is kept secret with the LU (lines 1 and 2).

As shown in Algorithm 4.2, each DO$_i$ encrypts its own private input n_i using the function *E()* and the public key *pk* (line 5). The encrypted input ε_i is then sent to the next DO$_{i+1}$ which adds ε_i with its own encrypted input ε_{i+1} (lines 6 to 20). The final encrypted sum sp_d is sent to back to the LU (line 13). Then the LU decrypts sp_d in function *D()* using *sk* to get the final sum $\mathcal{E}$ (line 22). The final sum $\mathcal{E}$ is sent to all DOs (line 23).

4.3.3.3 Salted Secure Summation Protocol

Salting is a technique that has been used to improve privacy against dictionary attacks on one-way hash functions where an additional string is concatenated with the value that is to be encrypted [189, 213]. As detailed in Algorithm 4.3, each DO$_i$ adds an additional random (salt) value r_i to its own private input n_i. Similar to ESS, the SSS protocol requires a trusted third party because individual salt values cannot be shared between the DOs to prevent collusion. Let us assume a LU is employed in this protocol.

Algorithm 4.3: Salted secure summation protocol (SSS)

```
Input  : $d$  - Number of DOs
         $n_i$ - The secret input of DO$_i$, $1 \le i \le d$
Output: $\mathcal{E}$  - Final sum where $\mathcal{E} \leftarrow \sum_1^d n_i$
1  for $i \in [1, 2, \cdots, d]$ do                         // All DOs follow the same steps
2      DO$_i$ computes a random salting value $r_i$
3      DO$_i$ sends $r_i$ to the LU
4      LU receives $r_i$
5      if $i == 1$ then                                    // Check if the current DO is DO$_1$
6          DO$_1$ computes partial sum $sp_1 \leftarrow n_1 + r_1$
7          DO$_1$ sends $sp_1$ to DO$_2$
8          DO$_2$ receives $sp_1$
9      end
10     else
11         DO$_i$ computes partial sum $sp_i \leftarrow sp_{i-1} + n_i + r_i$
12         if $i == d$ then                                // Check if the current DO is DO$_d$
13             DO$_d$ sends $sp_d$ to LU
14             LU receives $sp_d$
15         end
16         else
17             DO$_i$ sends $E_i$ to DO$_{i+1}$
18             DO$_{i+1}$ receives $E_i$
19         end
20     end
21 end
22 LU computes $r \leftarrow \sum_{i=1}^d r_i$                          // Add all random salt values
23 LU computes final sum $\mathcal{E} \leftarrow sp_d - r$    // Substract salt values from final partial sum
24 LU sends final sum $\mathcal{E}$ to DOs
25 DOs receive final sum $\mathcal{E}$
```

In lines 2 and 3, each DO_i starts by generating a random salting value r_i and sends it to the LU. Then DO_1 adds n_1 with r_1 and the resulting sum sp_1 is sent to DO_2 (lines 6 and 7). Following the same steps, each other DO_i (with $i > 1$) adds its own private input n_i and r_i to partial sum sp_{i-1} sent to it by the previous DO (lines 10 to 20). For example, DO_2 adds its private input n_2 and r_2 with sp_1 and the resulting sum sp_2 is sent to DO_3. Finally, DO_d sends the final partial sum sp_d to the LU (line 13). The LU computes the final sum $\mathcal{E}$ by subtracting the random salts of all DOs from sp_d (lines 22 and 23) and the resulting $\mathcal{E}$ is sent to all DOs (line 24).

4.3.3.4 Randomly-shared Secure Summation Protocol (RSS)

As detailed in Algorithm 4.4, the RSS protocol uses a set of d random shares of each private input value n_i to compute the final summation [201]. Similar to BSS, the RSS protocol can be used in linkage scenarios where no trusted third party is available. Each DO_i first computes d random shares (r_i^j, with $1 \leq j \leq d$) of its n_i by using the function *computeShares()* (line 2). Each share r_i^j is then sent to the corresponding DO_j (lines 3 to 7). Each DO_i then adds all the random shares it receives from the other DOs with the random share of its own n_i to calculate its partial sum sp_i (line 9). Each DO_i then sends sp_i to DO_1 (lines 10 to 13). DO_1 adds all sps to compute the final summation $\mathcal{E}$ (line 15). Finally, the computed $\mathcal{E}$ is sent to all other DOs (line 16).

Algorithm 4.4: Randomly-shared secure summation protocol (RSS)

```
Input  : $d$ - Number of DOs
         $n_i$ - The secret input of $\mathrm{DO}_i$, $1 \leq i \leq d$
Output: $\mathcal{E}$ - Final sum where $\mathcal{E} \leftarrow \sum_1^d n_i$
1  for $i \in [1, 2, \cdots, d]$ do                        // All DOs follow the same steps
2      $\mathrm{DO}_i$ computes a list of random shares $R_i \leftarrow computeShares(n_i, d)$, such that
       $\sum_{j=1}^d r_i^j \in R_i = n_i$
3      for $j \in [1, 2, \cdots, d]$ do                    // All DOs follow the same steps
4          if $i \neq j$ then                   // Send each share to its corresponding DO
5              $\mathrm{DO}_i$ sends a random share $r_i^j$ to $\mathrm{DO}_j$
6              $\mathrm{DO}_j$ receives $r_i^j$
7          end
8      end
9      $\mathrm{DO}_i$ computes $sp_i = \sum_{j=1}^d r_j^i$             // Add all the received shares
10     if $i \neq 1$ then                     // Check if the current DO is not DO$_1$
11         $\mathrm{DO}_i$ sends $sp_i$ to $\mathrm{DO}_1$
12         $\mathrm{DO}_1$ receives $sp_i$
13     end
14 end
15 $\mathrm{DO}_1$ computes final sum $\mathcal{E} \leftarrow \sum_{i=1}^d sp_i$       // Add all the partial sum values
16 $\mathrm{DO}_1$ sends final sum $\mathcal{E}$ to all other DOs
17 Other DOs receive final sum $\mathcal{E}$
```

4.3.3.5 Homomorphically Salted Secure Summation Protocol (HSS)

As detailed in Algorithm 4.5, we now propose a novel secure summation (HSS) protocol which is a hybrid approach between ESS and SSS, that is to combine the homomorphic encryption scheme with salting to perform secure communication between the DOs which is less vulnerable to collusion, as we discuss in the following subsection. Similar to both ESS and SSS, the HSS protocol employs a LU to compute the final summation value.

First, the LU generates the required public (*pk*) and private (*sk*) key pair for encryption and decryption of messages (line 1). Then the LU distributes *pk* to all DOs (line 2). As with SSS, each DO$_i$ first generates a random salting value r_i (line 5) and then adds its private input n_i to r_i to compute the partial sum value sp_i (line 6). Each DO$_i$ then encrypts r_i and sp_i using *pk* into r'_i and ε_i (lines 7 and 8).

Algorithm 4.5: Homomorphically salted secure summation protocol (HSS)

```
Input  : $d$ - Number of DOs
         $n_i$ - The secret input of DO$_i$, $1 \le i \le d$
Output: $\mathcal{E}$ - Final sum where $\mathcal{E} \leftarrow \sum_1^d n_i$
1  LU generates $pk$ and $sk$ of public and private key pair, respectively
2  LU sends $pk$ to all DOs
3  DOs receives $pk$
4  for $i \in [1, 2, \cdots, d]$ do                    // All DOs follow the same steps
5      DO$_i$ computes a random salting value $r_i$
6      $sp_i \leftarrow n_i + r_i$                     // Add the secret input and the salting value
7      $r'_i \leftarrow E(r_i, pk)$                    // Encrypt the salting value
8      $\varepsilon_i \leftarrow E(sp_i, pk)$          // Encrypt the partial sum value
9      if $i == 1$ then                                // Check if the current DO is DO$_1$
10         DO$_1$ sends $\varepsilon_1$ and $R'_i$ to DO$_2$
11         DO$_2$ receives $\varepsilon_1$ and $r'_i$
12     end
13     else
14         DO$_i$ computes partial sum $\varepsilon_i \leftarrow \varepsilon_{i-1} + \varepsilon_i$
15         DO$_i$ computes $r'_i \leftarrow r'_{i-1} + r'_i$
16         if $i == d$ then                            // Check if the current DO is DO$_d$
17             DO$_d$ sends $\varepsilon_d$ and $r'_d$ to LU
18             LU receives $\varepsilon_d$ and $r'_d$
19         end
20         else
21             DO$_i$ sends $\varepsilon_i$ and $r'_i$ to DO$_{i+1}$
22             DO$_{i+1}$ receives $\varepsilon_i$ and $r'_i$
23         end
24     end
25 end
26 LU computes final sum $\mathcal{E} \leftarrow D(\varepsilon_d, sk) - D(r'_d, sk)$    // Decrypt the received values
27 LU sends final sum $\mathcal{E}$ to DOs
28 DOs receive final sum $\mathcal{E}$
```

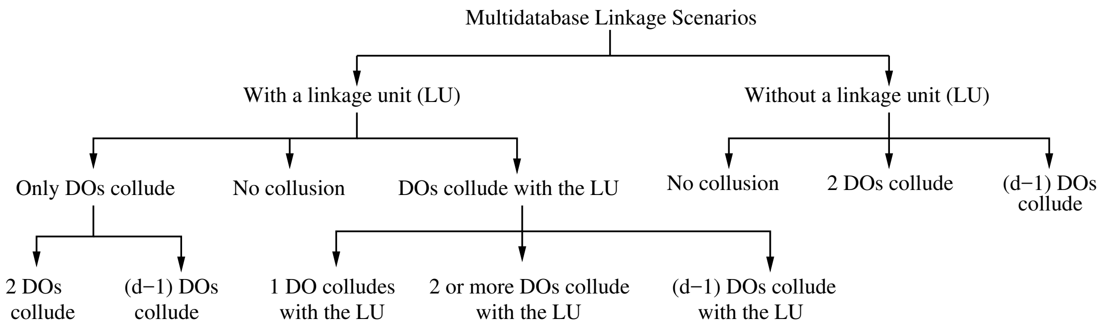


Figure 4.3: Collusion scenarios that are possible in MD-PPRL.

DO_1 then sends its encrypted values ε_1 and r'_1 to DO_2. DO_2 updates its encrypted results ε_2 and r'_2 by adding ε_1 and r'_1 as received from DO_1. DO_2 then sends the updated ε_2 and r'_2 to DO_3. All the other DOs follow the same process (lines 13 to 24), and finally DO_d sends ε_d and r'_d to the LU (line 17). The LU computes the final sum $\mathcal{E}$ by subtracting the decrypted sum of random salts from the decrypted final partial sum (line 26) and then sends $\mathcal{E}$ to all DOs (line 27).

4.3.3.6 Overall assessment of secure summation protocols

We now analyse the privacy of the five presented secure summation protocols with regard to different collusion scenarios and then evaluate their complexities in terms of computation and communication aspects with different numbers of DOs and input data sizes. As illustrated in Figure 1.3, we analyse the applicability of each protocol according to the two MD-PPRL models which are, (1) only DOs are participating and (2) a linkage unit (LU) is also involved in the linkage. We assume $d \geq 3$ DOs.

Privacy Analysis: As detailed in Section 2.4.5 collusion between the participating parties is a privacy risk in many real-world multi-party applications. Figure 4.3 shows the privacy of all presented protocols under different collusion scenarios.

As shown in Figure 4.3, for the linkage model without the LU, a group of DOs can collude with each other to identify the private input of one or more other DO(s). We consider three possible collusion scenarios under this model: (1) no collusion; (2) two DOs collude, where they aim to identify the private input of a third DO they communicate with; and (3) $d-1$ DOs collude with the aim of identifying the private input of the only non-colluding DO. This third scenario is the most challenging one.

For the linkage model with a LU, collusion can occur either between the DOs, or between the LU and one or more DOs. Apart from the collusion scenarios described above, DOs can collude with the LU in three different ways: (1) one DO colludes with the LU to identify the private input of another DO that it communicates with; (2) two or more DOs collude with the LU to learn the private input(s) of one or more other DO(s); and (3) $d-1$ DOs collude with the LU to identify the private input of the only non-colluding DO.

We now analyse the privacy of each presented secure summation protocol in terms of these collusion scenarios. The BSS protocol is most susceptible to collusion since the private input of any DO can be identified if its adjacent DOs collude with each other. For example, the BSS protocol with 3 DOs has a collusion risk if DO_1 and DO_3 collude with each other, which enables them to learn the private input of DO_2. Hence, the adding of a random value in BSS does not provide privacy against collusion for partially summed values computed in the intermediate protocol steps.

The ESS protocol uses a homomorphic encryption scheme to sum the private inputs of the participating DOs. However, ESS requires a LU to generate the keys (public and private) for the encryption and decryption of messages. Any collusion between DOs is not possible because the private key is only known to the LU. However, a collusion between DOs and the LU can compromise the privacy of the private input of another DO since each encrypted message with a partially summed result can be decrypted by the LU which can reveal the private input of a DO.

The SSS protocol allows each DO to generate their own random value (salt) to add to their private input value. Since the DOs that do not collude do not share their own random value with any other DO, collusion between DOs does not compromise the privacy of the private input values. However, the SSS protocol requires each individual salt value to be sent to the LU to compute the final summed value. Any collusion between DOs and the LU compromises the privacy of SSS as DOs can obtain the private value of another DO.

Compared to BSS, the RSS protocol is secure against collusion between DOs to learn another DO's private input since each DO keeps a random share of its own private value hidden from all the other DOs. However, since each DO sends its partially summed random shares to the first DO (DO_1), collusion between DO_1 and two or more other DOs can compromise the privacy of the protocol. For example if DO_1 colludes with $d-2$ other DOs, then DO_1 can identify the private value of the remaining DO by subtracting each shared private input from the final summation.

Our proposed HSS protocol improves overall privacy of the secure summation in two ways. First, each DO generates its own random salt value to be added to its private input. The salt value of a given DO is only shared with the other DOs and the LU in an encrypted form. This ensures even a collusion between $d-1$ DOs cannot deduce the random salt value of another DO. Second, each DO adds the encrypted summation value of its private and salt values to the encrypted partial sums received from the previous DO. Therefore, even if several DOs collude with the LU, neither the LU nor a DO can calculate the private value of any other DO.

However, if $d-1$ DOs collude with the LU, then the LU can calculate the private input of the remaining DO which will compromise the HSS protocol. This is still an open challenge for collusion scenarios where $d-1$ DOs collude. Another recently proposed secure summation protocol based on the ElGamal cryptosystem [65] by Mehnaz et al. [145] also provides privacy against $d-2$ DOs collusion scenarios, however, their approach requires longer runtime and complex computations compared to our HSS approach. We left the comparison of this novel protocol with our HSS protocol as future work.

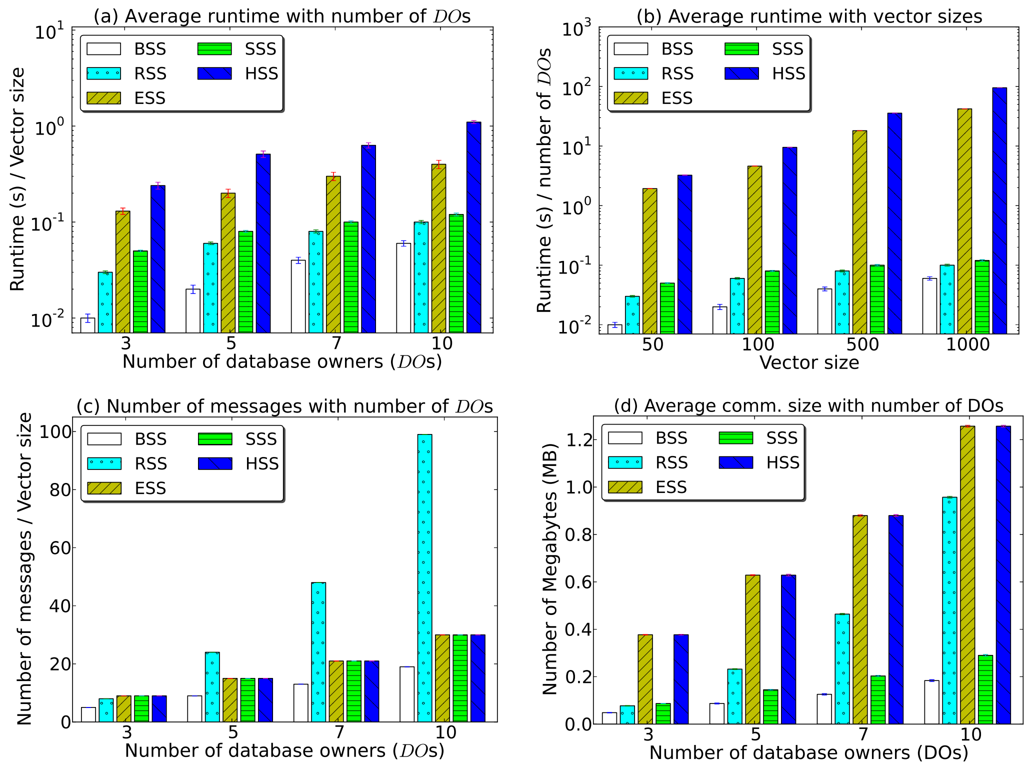


Figure 4.4: (a) Average runtime with different number of database owners (DOs), (b) average runtime for different vector sizes, (c) total number of messages communicated between DOs, and (d) average number of kilo bytes (KB) communicated in each protocol. Note that plots have different y-axis scales.

Complexity: To evaluate the computation and communication complexities of each presented secure summation protocol we experimented all secure summation protocols with 3, 5, 7, and 10 DOs. In each experiment we used a vector with 50, 100, 500, and 1,000 integer values as private input to each DO. For ESS and HSS we set the public and private key length to 128 bits. We ran all the experiments in a computational environment which will be detailed in Section 4.4.

As shown in Figure 4.4, we measured the average runtime of each protocol with different number of DOs and different input sizes, and the total number of messages communicated between the participants in each protocol. As Figure 4.4 (a) shows, as excepted ESS and HSS require longer average runtime per integer value communicated compared to the BSS, RSS, and SSS protocols. SSS is more efficient than ESS and HSS as both these protocols use homomorphic encryption and decryption functions on messages which are computationally expensive.

Figure 4.4 (b) shows that HSS requires longer runtime compared to the other protocols. However, as can be seen from Figure 4.4 (c), RSS requires a much larger number of messages to be communicated (as the number of DOs increases) because random shares have to be exchanged between each pair of DOs. In real-world sce-

Table 4.2: Categorisation of secure summation protocols in terms of privacy against collusion between different parties participating in the linkage.

Linkage models	Collusion scenarios					
	No collusion	2 DOs collude	$(d-1)$ DOs collude	1 DO colludes with the LU	2 or more DOs collude with the LU	$(d-1)$ DOs collude with the LU
Without a LU	BSS RSS	RSS	*No known protocol*	N/A	N/A	N/A
With a LU	ESS SSS HSS	ESS SSS HSS	ESS SSS HSS	ESS SSS HSS	HSS	*No known protocol*

narios RSS will possibly require longer runtime due to communication delays.

As shown in Figure 4.4 (d), ESS and HSS both send more as well as larger messages compared to BSS, RSS, and SSS since each message contains an encrypted value. However, the total message size of RSS increases quadratically with the number of DOs because of the pair-wise communication pattern of RSS. This potentially becomes expensive for large numbers of DOs.

Table 4.2 summarises the applicability of each of these secure summation protocols under different collusion scenarios. Overall, RSS and SSS are more suitable for linkage scenarios with and without a TP, respectively, in terms of efficiency. However, for scenarios where collusion is possible between the participating parties HSS is more suitable compared to all the other secure summation protocols as it provides the highest security.

4.4 Experimental Setup

In this thesis we evaluate our proposed framework with regard to the challenges of MD-PPRL (as described in Chapter 1). In Section 4.4.1 we provide a description of the datasets used to conduct the experimental evaluation in this thesis. Then, in Section 4.4.2, we describe the evaluation measures for assessing the three main properties of scalability, blocking quality, and privacy and we describe the implementation environment in Section 4.4.3.

4.4.1 Datasets

We use two real-world datasets to empirically evaluate and compare the performance of our proposed multidatabase PPRL framework. Table 4.3 provides an overview of the datasets we use in our experiments in Chapters 5 to 10.

1. **NC**: This is a large real-world database that contains voter registration data from the US state of North Carolina (NC)[1]. The **NC** dataset contains the names, addresses, and ages of over 7 million voters, as well as their voter registration numbers. This dataset has been used for the evaluation of various other RL and

[1]Available from: http://dl.ncsbe.gov/

Table 4.3: Datasets use in our experimental evaluations. 'No. of databases' is the number of individual data files (each considered as a different database) used in the experiments. 'Dataset Size (min-max)' is the minimum and maximum number of records in the dataset. 'Avg. overlap' is the average percentage of records matched across the dataset.

Datasets	No. of databases	Dataset size (min-max)	Avg. overlap	Provenance
NC-ORG	16	7,453,886 - 7,773,541	90%	Real
NC-CLN	16	5,614,747 - 7,453,886	90%	Real
NC-DRT	16	72,903 - 1,308,796	20%	Real
NC-SYN	10	5,000 - 1,000,000	50%	Synthetic
UK	6	17,033 - 31,059	5,000 records	Real

PPRL approaches [59, 90, 91, 109, 119, 212, 213, 216]. We are not aware of any other available large real-world datasets that contain records from more than two databases that could be used to evaluate techniques for MD-PPRL.

To allow the evaluation of our MD-PPRL approaches on different number of databases with different sizes we use 16 voter registration data files downloaded between February 2014 and December 2016 for our experiments. We assume each data file as a separate database that belongs to a different DO. The records in these data files can be categorised into one of three categories: (1) *exact matching*: those are records that are matching exactly with other records in a different data file over time, (2) *unique*: those are records that only appear in one data file, and (3) *updated*: those are records where at least one attribute value has changed across two data files collected at different points in time. We keep the identifiers of the records, which allows us to identify true and false matches and therefore calculate various blocking quality as discussed in Chapter 2 and Section 4.4.2.

In these **NC** datasets we use the *given name*, *surname*, *city*, and *Postcode* (*Zipcode*) attributes as blocking key attributes, as these are commonly used for RL [35, 174]. We use four different variations of the **NC** datasets in the experiments as described below.

- **NC-ORG**: This dataset contains 16 original voter registration data files each containing between 7,453,886 and 7,773,541 records. Each data file of **NC-ORG** contains all three types of *exact matching*, *unique*, and *updated* records.
- **NC-CLN**: From the **NC** dataset we extract *unique* and *exact matching* records in each data file into a separate dataset **NC-CLN** (clean), where each data file contains between 5,614,747 and 7,453,886 records and in each pair of data files more than 90% of records are exact matches. Due to the skewness of *exact matching* records we use the **NC-CLN** dataset only to evaluate the scalability of our blocking techniques.

- **NC-DRT**: We extract *unique* and *updated* records from each **NC** data file where the data files in **NC-DRT** (dirty) contain between 72,903 and 1,308, 796 records. Since each update in an attribute value across different points in time is considered as a modification in a record, these data files are used to evaluate the effect of real-world dirty data on the blocking quality of our approaches.

- **NC-SYN**: In order to evaluate our approach with different levels of data quality, we use a synthetic dataset (**NC-SYN**) that is created by extracting records from the original **NC** dataset. We used and modified a recently proposed personal data generation and corruption tool (GeCo)[2] to generate these synthetic datasets [40, 204].

  This dataset contains subsets including 3, 5, 7 and 10 databases, where each subset contains datasets with 5,000, 10,000, 50,000, 100,000, 500,000 and 1,000,000 records. In each of these subsets, 50% of records were matches, i.e. half of all records occur in the subsets of all databases.

  We also created groups of datasets where we included a varying number of corrupted records into the sets of overlapping records (ranging from 0% to 80% corruption, in 20% steps). We applied various corruption functions in different numbers (ranging from 1 to 3) to randomly selected attribute values. This allows us to investigate how our techniques can handle dirty data. We applied various corruption functions, including character edit operations (insertions, deletions, substitutions, and transpositions), and optical character recognition and phonetic modifications based on look-up tables and corruption rules [40].

  As a result, in these synthetic datasets a certain percentage of records in the overlap were modified for randomly selected databases. Therefore, some of these records are *exact matching* across some databases, while in other databases these records cannot be exactly matched. This simulates for example the situation where three out of five hospitals have the correct and complete contact details (like name and address) of a certain patient, while in the fourth and fifth hospital some of the details of the same patient are changed, contain errors, or are missing.

2. **UK**: This dataset, as used in and provided by [77], consists of census records collected from the years 1851 to 1901 in 10 year intervals for the town of Rawtenstall and surrounds in the United Kingdom (UK). It contains approximately 150,000 individual records of 32,000 households. Nearly 5,000 records have been manually linked by domain experts providing partial gold standard data for testing. We consider data for each census year as a different database which allows us to evaluate our approaches with six different databases.

[2]Available online: http://dmm.anu.edu.au/geco [204]

4.4.2 Evaluation Measures

For the empirical evaluation of our proposed framework we consider the three properties of privacy, scalability, and blocking quality of PPRL. As we will describe next, scalability and blocking quality are measured based on the candidate record tuples (CBTs) generated by our framework.

4.4.2.1 Privacy Measures

To evaluate the privacy of different techniques proposed in our framework we assess the risk of disclosure by calculating the probability that an attacker can correctly identify a value in a database (as described in Section 2.5 on page 38). To calculate the risk of disclosure of each attribute value (BKV) an attacker can use a publicly available database (global database) to conduct a frequency analysis on the attribute values (as described in Section 2.4).

To evaluate the privacy under the worst case scenario, we assume a publicly available database $\mathbb{G}$ which can be considered to be equivalent to the linked database $\mathbf{D}$, even though this might not be practical in many real-world situation. We consider the assumption $\mathbb{G} \equiv \mathbf{D}$ since the calculation of disclosure risks using an external global database depends on the choice of the global database [54]. In addition, an external global dataset might not be available for privacy evaluation. Hence, conducting a privacy attack using the database $\mathbf{D}$ as $\mathbb{G}$ provides the highest possible risk of disclosure of a real-world PPRL protocol if an attacker has access to such database.

As described in Section 2.5, we use the normalised probability of suspicion (PS) and disclosure risk (DR) measures to evaluate the overall privacy of each technique in our framework. Since MD-PPRL involves several DOs ($d \geq 3$) we compute the DR values for each database to be linked separately and compute the average to assess the overall privacy of our framework. The set of DR measures that are used in our framework is given below.

$$Average\ maximum\ disclosure\ risk\ (Avg.DR_{Max}) = \frac{\sum_{i=1}^{d} DR_{Max}(i)}{d} \tag{4.4}$$

$$Average\ mean\ disclosure\ risk\ (Avg.DR_{Mean}) = \frac{\sum_{i=1}^{d} DR_{Mean}(i)}{d} \tag{4.5}$$

$$Average\ median\ disclosure\ risk\ (Avg.DR_{Med}) = \frac{\sum_{i=1}^{d} DR_{Med}(i)}{d} \tag{4.6}$$

$$Average\ marketer\ disclosure\ risk\ (Avg.DR_{Mkt}) = \frac{\sum_{i=1}^{d} DR_{Mkt}(i)}{d} \tag{4.7}$$

We also measure the corresponding minimum, maximum, and median of these disclosure risk values from d databases as well.

4.4.2.2 Scalability Measures

Scalability is the second property that needs to be measured in a multidatabase linkage. Scalability of a blocking technique depends on the complexity of the algorithms used which is in general measured using the big-O notation [7]. We measure the complexity of our approaches in terms of the number of record pair comparisons that are generated by blocking (records pairs that need to be compared in a given set of *CRT*s).

As detailed in Chapter 2, blocking reduces the number of record pair comparisons by removing those pairs that are unlikely to refer to matches without comparing them in detail in the comparison step of both RL and PPRL. As we detailed in Section 1.2, the naïve pairwise comparison of multiple databases is of exponential and quadratic complexities in the number and size of the databases to be linked, respectively.

In a MDRL context, separate pairwise matchings of records do not guarantee the transitivity of the linkage decisions and thus require calculating similarities for each record pair [73]. The records in each *CRT* need to be compared pairwise to identify each *CRT* as a match or a non-match [183, 184].

For example, let us assume a $CRT = \langle r^1_A, r^1_B, r^1_C \rangle$ of databases $\mathbf{D}_A, \mathbf{D}_B$, and $\mathbf{D}_C$, where $r^1_A \in \mathbf{D}_A$, $r^1_B \in \mathbf{D}_B$, and $r^1_C \in \mathbf{D}_C$. The comparison of these three records requires to solve five linkage possibilities (clusterings): (1) r^1_A and r^1_B refer to the same entity but r^1_C refers to a different entity, (2) r^1_A and r^1_C refer to the same entity but r^1_B refers to a different entity, (3) r^1_B and r^1_C refer to the same entity but r^1_A refers to a different entity, (4) r^1_A, r^1_B, and r^1_C refer to three different entities, and (5) r^1_A, r^1_B, and r^1_C refer to the same entity.

However, due to errors in attribute values of these records only using the calculated similarities of record pairs (r^1_A, r^1_B) and (r^1_B, r^1_C) can not resolve all these linkage possibilities, because r^1_C might not be matched with r^1_A. Therefore, the linkage between r^1_A, r^1_B, and also r^1_C requires r^1_A to be compared with r^1_C. Hence, each *CRT* with records from d databases requires $\frac{d \times (d-1)}{2}$ record pair comparisons. We now define scalability measures for the MDRL context.

As described in Section 2.5, reduction ratio (RR) is a standard measure used in record linkage to assess the efficiency of a blocking technique [36]. RR measures how many candidate record pairs a blocking technique is able to reduce compared to all possible record pairs. Assuming each database contains n_r records, the total comparison space results in $n_{total} = (n_r)^d \times \frac{d \times (d-1)}{2}$ record pair comparisons for the naïve pairwise comparison of d databases. We calculate the RR of blocking of d databases as:

$$RR = \frac{number\ of\ candidate\ record\ tuples \times \frac{d \times (d-1)}{2}}{n_{total}}, \tag{4.8}$$

where each candidate record tuple is of size d. This RR can be further simplified as:

$$RR = \frac{number\ of\ candidate\ record\ tuples}{(n_r)^d}, \tag{4.9}$$

Table 4.4: Different linkage possibilities of a $CRT = \langle r^1_A, r^1_B, r^1_C \rangle$ with $r^1_A \in \mathbf{D}_A$, $r^1_B \in \mathbf{D}_B$, and $r^1_C \in \mathbf{D}_C$. The match and non-match status are considered for the linkage across all three databases $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}_C$ (i.e. no subset matching is considered). In this example, $=$ and $\neq$ represent a match and non-match, respectively.

Clustering		Linkage Possibility	Match or Non-match
(1)	$\langle r^1_A, r^1_B, r^1_C \rangle$	$(r^1_A = r^1_B)$, $(r^1_A = r^1_C)$, $(r^1_B = r^1_C)$	Match
(2)	$\langle (r^1_A, r^1_B), r^1_C \rangle$	$(r^1_A = r^1_B)$, $(r^1_A \neq r^1_C)$, $(r^1_B \neq r^1_C)$	Non-match
(3)	$\langle r^1_A, (r^1_B, r^1_C) \rangle$	$(r^1_A \neq r^1_B)$, $(r^1_A \neq r^1_C)$, $(r^1_B = r^1_C)$	Non-match
(4)	$\langle (r^1_A, r^1_C), r^1_B \rangle$	$(r^1_A \neq r^1_B)$, $(r^1_A = r^1_C)$, $(r^1_B \neq r^1_C)$	Non-match
(5)	$\langle (r^1_A), (r^1_B), (r^1_C) \rangle$	$(r^1_A \neq r^1_B)$, $(r^1_A \neq r^1_C)$, $(r^1_B \neq r^1_C)$	Non-match

To practically evaluate the efficiency and scalability of a blocking technique that is used in our framework, we use several measures that are dependent on the computing platform and the networking infrastructure used. To measure the scalability of each blocking technique in terms of size of the databases we compute the average runtime per DO that is required to perform blocking on a database with n_r records (assuming each DO performs blocking upon its database in its own computing environment). We also use the total runtime of the entire protocol (summation of runtime of all steps in a protocol including the communication steps) to measure the scalability in terms of the number of databases to be linked. We also measure the memory space required to perform the blocking, and the size and the number of messages communicated between different parties (as defined in Section 2.4) that participate in a linkage.

4.4.2.3 Quality Measures

Quality of blocking measures the effectiveness of the grouping of records into blocks. In practice, the quality of blocking is assessed based on the available truth data which is often difficult because no truth data with known match status are available in many real-world applications [38].

In a multidatabase context, to consider a *CRT* of size d as a true match for the linkage of corresponding d databases, all record pairs of the *CRT* need to be matched. For example, to consider a $CRT = \langle r^1_A, r^1_B, r^1_C \rangle$ as a true match it requires the record pairs (r^1_A, r^1_B), (r^1_A, r^1_C), and (r^1_B, r^1_C) to be matched and all the other linkage possibilities ((2) to (5) as discussed in Table 4.4) between these three records are considered as non-matches for the linkage of three databases. Table 4.4 summarises the different linkage possibilities of this *CRT* for the linkage of databases $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}_C$ and their match and non-match status.

Similarly, for subgroups of databases of different sizes that need to be linked, the matches between subsets of record pairs are considered. For example, the record clusterings (2), (3), and (4) shown in Table 4.4 are considered as matches for linkages between databases ($\mathbf{D}_A$,$\mathbf{D}_B$), ($\mathbf{D}_B$,$\mathbf{D}_C$), and ($\mathbf{D}_A$,$\mathbf{D}_C$), respectively. Similarly as mentioned above in Section 4.4.2, due to errors and variations in attribute values, out of the record pairs (r_A^1, r_B^1), (r_A^1, r_C^1), and (r_B^1, r_C^1) of $CRT = \langle r_A^1, r_B^1, r_C^1 \rangle$, only ($r_A^1$, r_B^1) and (r_A^1, r_C^1) could potentially be matches which makes the given CRT a match for the only linkage of subgroups ($\mathbf{D}_A$,$\mathbf{D}_B$) and ($\mathbf{D}_A$,$\mathbf{D}_C$) of databases $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}_C$.

In both RL and PPRL the quality of blocking is measured based on the classification of the number of true matches (TM), false matches (FM), false non-matches (FN), and true non-matches (TN), as well as true matches included in the candidate record pairs generated by blocking (BM), and true non-matches included in the candidate record pairs (BN), as described in Section 2.5.

We use the measures pairs completeness (PC) and pairs quality (PQ) for measuring the quality of blocking [35]. We adapted the TM, FM, FN, and TN count as the number of true matching, false matching, false non-matching, true non-matching record tuples, and BM and BN as true matching and true non-matching CRT, as discussed above, included in the generated CBTs, respectively, to facilitate the assessment of blocking quality in MDRL and MD-PPRL contexts. We define the PC and PQ for blocking of d databases as:

$$PC = \frac{BM}{TM + FN} \tag{4.10}$$

$$PQ = \frac{BM}{BM + BN} \tag{4.11}$$

As detailed in Section 2.5, F-measure (FM) is commonly used in RL applications to measure the quality of classification [35, 229]. We adapted F-measure to measure the effectiveness of blocking by computing the harmonic mean between PC and RR which is calculated as:

$$FM = 2 \times \left(\frac{PC \times RR}{PC + RR} \right). \tag{4.12}$$

4.4.3 Implementation Environment

We implemented all our proposed approaches using the Python programming language [142] (version 2.7.3) since Python is flexible and supportive for writing programmes and scripts for rapid prototype development, and is ideally suitable for iterative development of prototypes. We ran all our experiments on a server with 64-bit Intel Xeon (2.4 GHz) CPUs, 128 GBytes of main memory, 4 TBytes of disk space, and running Ubuntu 14.04.

To simulate the communication between the database owners and other parties involved in a linkage, we created separate directories for each party that participates in the linkage to store its data. These directories are used to read and write data into files to represent communication of messages between corresponding parties in the linkage. An advantage of this approach is that during the execution of communication steps no additional memory is needed to keep the communicated data. This mechanism enables us to simulate RL and PPRL among multiple parties on the same machine without requiring different individual machines.

4.5 Cryptanalysis based Privacy Evaluation for MD-PPRL

As detailed in Section 2.4 on page 33, PPRL techniques are susceptible to privacy attacks. In such attacks, an adversary can try to link records from an encoded (masked) database to records in a global (publicly available) database in order to re-identify records and/or attribute values in the encoded database (i.e. match encoded records, such as Bloom filters, to plain text records). We use a recently proposed re-identification (cryptanalysis) attack method [39] to evaluate the privacy of the proposed blocking techniques of our framework.

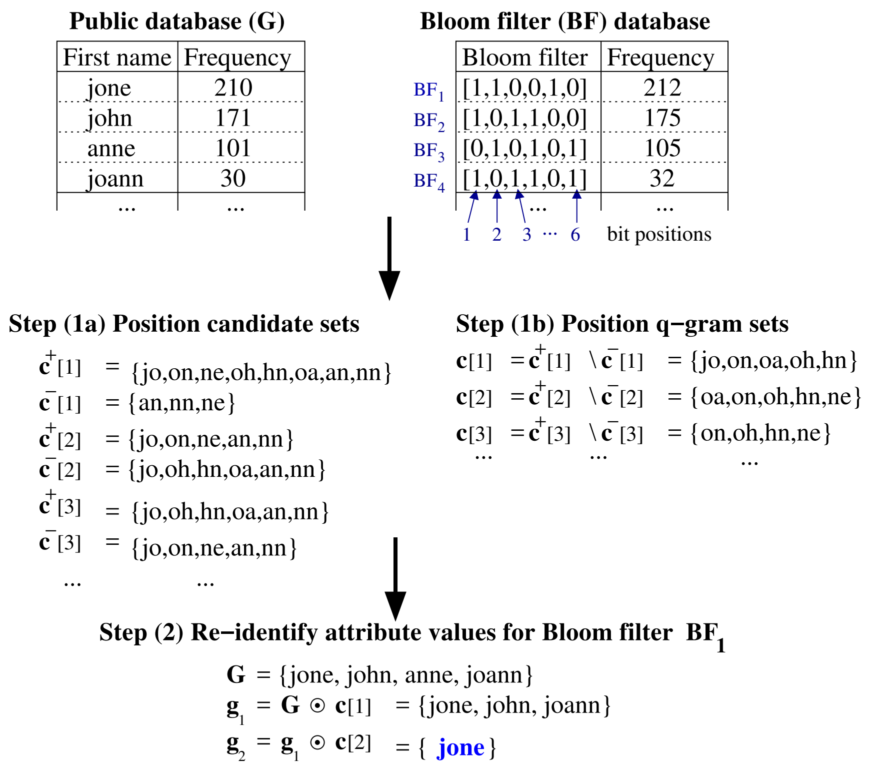
In our privacy evaluation we consider possible privacy attacks by parties involved in the linkage. A privacy attack by a participating party provides the worst case scenario of a privacy leakage, since a participating party has likely access to more information than an external adversary, including knowledge about the PPRL protocol used, as well as the encoding methods and parameter values of the linkage techniques and encoding algorithms used.

We use the attack method proposed by Christen et al. [39] to simulate a frequency attack on the proposed Bloom filter (BF) based blocking techniques (as described in Chapters 5, 6, and 7) used in our framework. As described in Section 2.4, the proposed frequency attack method uses frequency counts and patterns in a set of BFs to iteratively map bit patterns to known attribute values, aiming to re-identify the encoded sensitive attribute values. This attack method does not require any assumption on the BF parameters used when sensitive attribute values were encoded which is appropriate in real-world scenarios.

The overview of this attack method is shown in Figure 4.5. In this method the attacker has access to a set of encoded BFs, $\mathbf{B}$, and their frequencies. Since the attacker does not know anything about the set of parameters that has been used in the encoding process, he can sample attribute values from a publicly available database $\mathbb{G}$ (such as a telephone directory) and select a set of frequent values, $\mathbf{V}$, from an attribute that has a frequency distribution similar to the distribution of $\mathbf{B}$. As shown by Schnell and Borgs [193], the attacker can guess which attribute(s) has/have been encoded based on the frequency distribution of the number of 1-bits (Hamming weights) in $\mathbf{B}$, because different attributes have distinctive frequency distributions.

As shown in Figure 4.5, in step (1a), the attacker first aligns BFs and attribute values according to their frequencies and selects the set of most frequent values to analyse further in the attack. Next he analyses each bit position in these selected BFs and generates the sets of possible q-grams that can be assigned to each of these bit positions. As shown in Figure 4.5, for each bit position i in the BFs, all corresponding q-grams of the attribute values that have bit i set to 1 are added to the set $\mathbf{c}^+[i]$. The set of q-grams of all attribute values with a value of 0 at bit position i are added to the set $\mathbf{c}^-[i]$ of *not* possible q-grams for that position, because a 0-bit means no q-gram of an attribute value could have been mapped to position i.

In step (1b), for each position i an attacker obtains the set $\mathbf{c}[i] = \mathbf{c}^+[i] \setminus \mathbf{c}^-[i]$ of q-grams that potentially could have been hashed to position i. Based on the attribute values from $\mathbb{G}$ the attacker can try to identify which BF possibly encodes which value



Public database (G)

First name	Frequency
jone	210
john	171
anne	101
joann	30
...	...

Bloom filter (BF) database

	Bloom filter	Frequency
BF_1	[1,1,0,0,1,0]	212
BF_2	[1,0,1,1,0,0]	175
BF_3	[0,1,0,1,0,1]	105
BF_4	[1,0,1,1,0,1]	32
	...	...

Step (1a) Position candidate sets

$\mathbf{c}^+[1]$ = {jo,on,ne,oh,hn,oa,an,nn}
$\mathbf{c}^-[1]$ = {an,nn,ne}
$\mathbf{c}^+[2]$ = {jo,on,ne,an,nn}
$\mathbf{c}^-[2]$ = {jo,oh,hn,oa,an,nn}
$\mathbf{c}^+[3]$ = {jo,oh,hn,oa,an,nn}
$\mathbf{c}^-[3]$ = {jo,on,ne,an,nn}
... ...

Step (1b) Position q−gram sets

$\mathbf{c}[1] = \mathbf{c}^+[1] \setminus \mathbf{c}^-[1]$ = {jo,on,oa,oh,hn}
$\mathbf{c}[2] = \mathbf{c}^+[2] \setminus \mathbf{c}^-[2]$ = {oa,on,oh,hn,ne}
$\mathbf{c}[3] = \mathbf{c}^+[3] \setminus \mathbf{c}^-[3]$ = {on,oh,hn,ne}
... ...

Step (2) Re−identify attribute values for Bloom filter BF_1

$\mathbf{G}$ = {jone, john, anne, joann}
$\mathbf{g}_1 = \mathbf{G} \odot \mathbf{c}[1]$ = {jone, john, joann}
$\mathbf{g}_2 = \mathbf{g}_1 \odot \mathbf{c}[2]$ = { **jone** }

Figure 4.5: Outline of the cryptanalysis attack (adapted from [39]).

in $\mathbb{G}$ by considering the sets of possible q-grams in list $\mathbf{C} = [\mathbf{c}[1], \ldots, \mathbf{c}[l_{bf}]]$, where l_{bf} is the length of a BF.

As shown in Figure. 4.5, in step (2) an attacker can analyse each BF in $\mathbf{B}$ and remove those attribute values from $\mathbb{G}$ that are not possible matches according to $\mathbf{C}$ because they do not contain any q-grams that would have been hashed to a certain 1-bit. For example, for the most frequent BF_1 in Figure. 4.5, *jone* is a possible attribute value because the 1-bits in positions 1 and 2 contain the set of q-grams of *jo*, *on*, and *ne*.

4.6 Summary

In this chapter we have introduced our novel blocking framework that can be used to block multiple databases efficiently and effectively. The proposed framework contains three layers each contributing to a specific aspect in blocking in a multi-database context. In our framework the first layer generates blocks for each database that needs to be linked, the second layer identifies the blocks that need to be compared across different subgroup combinations of these databases, and the third layer reduces the record comparison space by removing redundant record comparisons. Each layer comprises different techniques that can be used to facilitate the linkage between databases according to user requirements.

We also provided details on the building blocks (components, data structures, and communication methods) that are used in our approaches followed by a detailed description of the experimental setup including the datasets, evaluation measures, and computation environment that we use in the experiments. Finally, we provided a detailed description about a cryptanalysis method that we use to evaluate the privacy of our proposed techniques. Each of the following chapters describes in detail one of our proposed novel MD-PPRL blocking techniques that are used in our framework. We will then empirically evaluate these techniques in Chapter 10 using the experimental setup presented in this chapter.

Chapter 5

Tree based Blocking for Multidatabase Privacy-preserving Record Linkage

Addressing the scalability challenge in MD-PPRL is a primary dimension we consider in our research. As mentioned in Chapter 1, the record pair comparison space grows exponentially with an increase in the number of databases. In this chapter we propose an efficient blocking technique that can be used in our blocking framework. In Section 5.1 we introduce the aim of our blocking technique and in Section 5.2 we provide an overview of our approach. As detailed in Section 5.3, our proposed approach minimises the multidatabase record comparison space by effectively grouping records into blocks using Bloom filters and a tree data structure. We analyse our blocking technique in terms of complexity, blocking quality, and privacy in Section 5.4, and empirically validate our analysis with a set of experiments as detailed in Section 5.5. Finally, we summarise our findings in Section 5.6.

5.1 Introduction

Scalability is one prominent challenge that needs to be tackled by any linkage application. As detailed in Chapter 1, when the number of databases to be linked is increasing the record comparison space grows exponentially which makes the MD-PPRL process significantly more challenging. Blocking (indexing) has been used for many years in linking of two databases. Blocking reduces the large number of potential comparisons by removing as many record pairs as possible that correspond to non-matches. As a result, blocking decreases the amount of computational and communication efforts required for the comparison of large databases. As surveyed by Christen [36], Papakasidis et al. [165], and Steorts et al. [199], most existing blocking techniques are only considering the linkage of two databases. This provides us with the motivation to develop blocking techniques for MD-PPRL that scale with an increase in the number of databases to be linked.

Table 5.1: Notation and terminology used in this chapter

BF	Inverted index of Bloom filters
BT	Block tree structure
D	A database
A	Set of blocking attributes
B_0, B_1	Block of Bloom filters having 0 and 1 in a given bit position
$\mathcal{H}$	Set of hash functions
Q	A queue data structure
$R, R.a, R.id$	A record, record attribute value, and a record identifier
S	A set of q-grams
$V_R, V_{\mathbf{R}}, V_G$	Ratio vector, random vector, and a global summed ratio vector
bf	A Bloom filter in **BF**
b_{min}, b_{max}	Minimum block size and maximum block size
d	Number of database owners (or databases)
f_{ij}	Ratio of bit position j of DO$_i$
fp	False positive rate
$h(\cdot)$	A hash function used for mapping a value into a Bloom Filter
j_G	Best splitting bit position
n_h, n_q, n_r	Number of hash functions, q-grams, and records
l_q, l_{bf}	Length of a q-gram and a Bloom filter
o_{ij}	Number of 1's in bit position j for DO$_i$
q	A q-gram
v, v_0, v_1	A node, node with B_0, and a node with B_1
BF	Bloom filter
BK	Blocking key attribute
BKV	Blocking key value
DO	Database owner
PPRL	Privacy-preserving record linkage
QID	Quasi-identifier
MD-PPRL	Multidatabase PPRL
SBT	Single bit tree

In this chapter we propose a blocking technique for MD-PPRL that can provide improved scalability, blocking quality, and privacy, which are important factors for any practical PPRL applications. Our approach is based on an efficient tree data structure to group records into blocks. The primary aim of our blocking approach is to allow the database owners (DOs) to block their databases collaboratively without using a linkage unit (LU). As we will describe in Section 5.4, this improves the privacy of our approach since the participating DOs are less likely to collude [216]. This makes our approach suitable for practical linkage scenarios where no LU is available or can be used to conduct the linkage. We also use a secure multi-party computation (SMC) protocol, as described in Section 4.3 to perform computations securely across the DOs. The following subsections provide the proposed approaches in more detail. Table 5.1 provides the notation and terminology we use in this chapter.

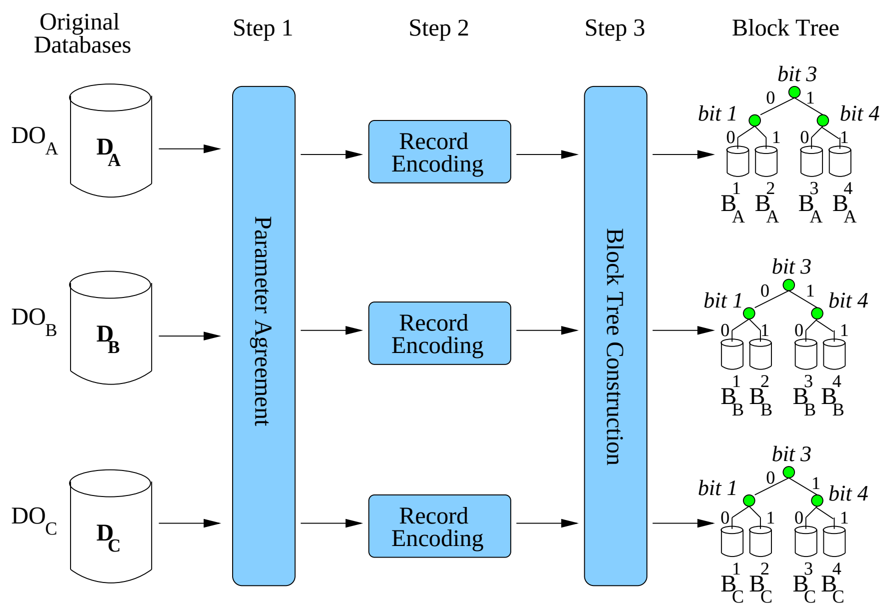


Figure 5.1: A high-level overview of our approach. $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}_C$ represent the databases held by the database owners DO_A, DO_B, and DO_C, respectively. In step 1 all database owners need to agree on the parameter settings. In step 2 the database owners independently encode their databases. In step 3 these encoded databases are blocked by the database owners as we describe in Section 5.3. The output of step 3 is a block tree structure for each database where the blocks are represented as leaves. As we describe in Section 5.3, the blocks of each corresponding leaf in the block trees need to be compared because each block tree is constructed in the same manner. These corresponding blocks from different databases can then be compared using a private comparison and classification technique [216]. For example, blocks $\langle B^1_A, B^1_B, B^1_C \rangle$, $\langle B^2_A, B^2_B, B^2_C \rangle$, $\langle B^3_A, B^3_B, B^3_C \rangle$, and $\langle B^4_A, B^4_B, B^4_C \rangle$ are being compared because they are in the leaves with the same path.

5.2 Overview of our approach

One major problem in currently available blocking solutions is that they do not provide enough control to the DOs over the block generation process. In the comparison step the blocks that contain a larger number of records will require more computational time because more records need to be compared as we explained in Section 4.4.2. On the other hand, blocks generated with a small number of records are vulnerable to privacy attacks as we will explain in more detail in Section 5.4. To overcome these problems, we suggest a parameterised blocking approach, where the user can control the size of the blocks that will be generated. A high-level overview of our approach is shown in Figure 5.1.

As illustrated in this figure, our approach has three main steps. In the first step the DOs need to agree upon the parameters used for the blocking. Once the blocking parameters are agreed upon, each DO encodes each of their records into Bloom filters (BFs), as explained in Section 4.3 on page 68, to mask its original record values from the other DOs in step 2. This record encoding process improves the privacy of the individual attribute values since the computations in the next phase are only performed on these encoded values rather than on the original values. This record encoding process is performed by each DO independently based upon the agreed parameter values.

As illustrated in Figure 5.1, in the third step of our approach, each DO constructs a balanced binary tree data structure (which we named as *block tree*) where the leaf nodes contain the blocks generated for their database. As we reviewed in Chapter 3, Schnell was the first to use a binary tree data structure as a blocking mechanism for two database PPRL [188]. In our approach the tree construction process is performed collaboratively where all DOs securely compute how the records in their respective databases are arranged into blocks by using a secure communication protocol. Once the blocking is completed each participating DO will have the same block tree structure where the records in the respective blocks can be compared using any private comparison and classification technique [212].

We assume d DOs in our approach each with a database, and $d \geq 3$. We also assume a set A of quasi-identifiers [94], such as first name, last name, address details, etc., are common to all the databases. These attributes in A are used as blocking key (BK) attributes to group similar records into blocks.

5.3 Tree Based Scalable Blocking Approach

In this section we describe each phase of our approach in detail.

1. **Parameter Agreement**

   In step 1 of our approach, all DOs agree upon the set of parameters to be used for the block generation process. For the encoding of records in step 2, each DO needs to agree upon the length of the BF, l_{bf}, the length (in characters) of q-grams, l_q, the number n_h of hash functions ($\mathcal{H}$), and the set A of attributes to be used as BKs. To control the size of the blocks generated by our approach we introduce the two parameters *minimum block size*, b_{min}, and *maximum block size*, b_{max}. These parameters specify the minimum and maximum number of records that need to be included in a block, respectively. The DOs need to agree upon the b_{min} and b_{max} parameter values before starting to construct their tree structures.

2. **Record Encoding**

   In the second step of our approach, each DO$_i$ encodes the records in its database $\mathbf{D}_i$ into BFs as detailed in Algorithm 5.1. In line 2, each DO$_i$ iterates over its database and each record R is encoded into a BF. The function *genBloomFilter()* creates a bit vector bf of length l_{bf} with all bits set to 0 (line 3).

   In lines 4 and 5, each blocking key value (BKV) in $R.a$ for each BK $a \in A$ is converted into a set S of q-grams by using the function *genQGrams()*. Each

Algorithm 5.1: Bloom filter generation by DO_i

```
Input  : D_i  - Database belonging to DO_i
         A    - Set of selected BK attributes
         l_q  - Character length of a q-gram
         l_bf - Length of a Bloom filter
         H    - Set of hash functions, where |H| = n_h
Output: BF_i - Inverted index of Bloom filters
BF_i ← {}                                  // Initialisation of the inverted index
foreach R ∈ D_i do                          // Iterate over each record
    S ← ∅, bf ← genBloomFilter(l_bf)        // Generates an empty Bloom filter
    foreach a ∈ A do                        // Iterate over each blocking attribute
        S.add(genQGrams(R.a, l_q))          // Generates set of q-grams
        foreach q ∈ S do                    // Iterate over each q-gram
            foreach h ∈ H do                // Iterate over each hash function
                j ← h(q)                    // Get the bit position
                bf[j] ← 1                   // Set the bit position to 1
            end
        end
    end
    BF_i[R.id] ← bf                         // Add the generated Bloom filter to BF_i
end
return BF_i
```

q-gram in a set S is then hashed using the set $\mathcal{H}$ of hash functions, and the respective index positions in bf are set to 1 (lines 6 to 9). Finally, each BF bf is added to an inverted index data structure $\mathbf{BF}_i$ using its record identifier ($R.id$) as a key (line 13). The $\mathbf{BF}_i$ is used in the next step of our approach to construct the block tree structure.

3. **Block Tree Construction**

   In the third step of our approach, each DO constructs its block tree based on a single-bit tree data structure by using the generated $\mathbf{BF}_i$ from their database. A single-bit tree (SBT) is a binary tree data structure that can be used to store information about a set of bit vectors. SBT was proposed by Kristensen et al. [129] to efficiently find similar chemical fingerprints in a database (as was described in Chapter 3 on page 48).

   The construction of the tree starts from the root node, where all bit vectors are assigned to the root. At each node in the tree a position i in the bit vector is chosen to best split all the BFs of the node into two partitions of ideally equal size, which in turn will keep the tree as balanced as possible. All bit vectors with a 0 at position i are stored in the left sub-tree while all bit vectors with a 1 at position i are stored in the right sub-tree. This division is repeated recursively until all the bit vectors in a given node are the same, or all the bit positions have been used for the construction, or all blocks in leaf nodes are within the required size limit. Figure 5.2 shows an example of a SBT.

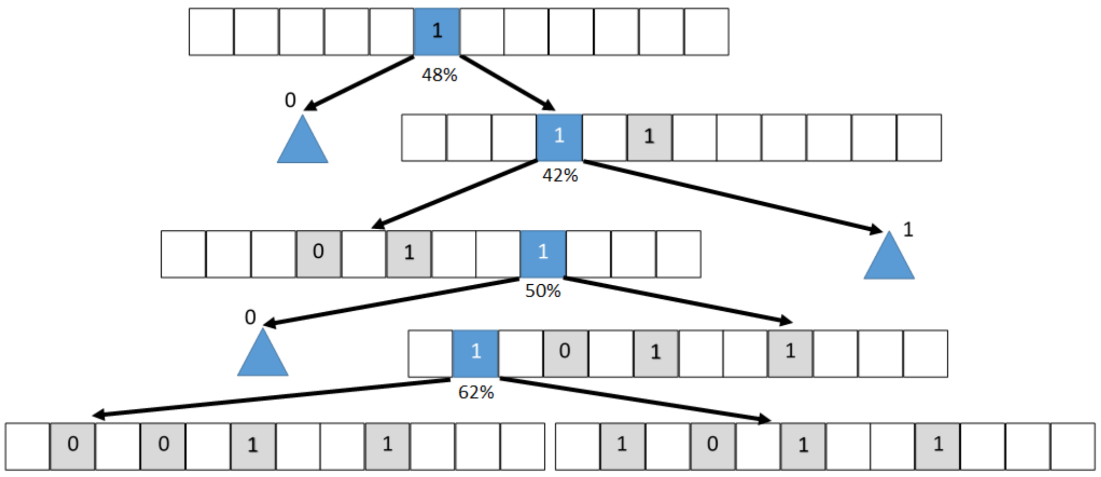


Figure 5.2: An illustrative example of a single-bit tree. In this example, the blue squares represent the bit positions chosen for the given node, while the light gray squares mark bit positions chosen at an ancestor. The blue triangles represent sub-trees that are not shown. The percentage value under each chosen bit position shows the number of bit vectors that contain a 0 or 1 in the selected position out of all the bit vectors processed in a node.

It is not directly apparent how best to choose which bit position to split the data on at a given node when building the tree structure. The selection of the best splitting bit position requires information about all the bit vectors (BFs) held by a given node when building the tree. This requires all the BFs assigned to a given node to be inspected, and then to select a bit position which contains 0 in as close to half of all the BFs as possible and 1 in the others. The construction of a SBT for an individual DO is shown in Algorithm 5.2.

In the beginning of the tree construction, a root node is created and the full inverted index $\mathbf{BF}_i$ is assigned to this root node as its data (line 1). A queue Q is created to hold the nodes, where initially the root node is assigned into the queue (line 2). We designed the tree construction to compute nodes in an iterative manner rather than recursively which is more efficient in processing nodes according to a set of experiments. In each iteration the node at the beginning of Q is processed. The processing of a node can be split into three sub-steps, which are:

3.a Perform secure summation to find the best splitting bit position.

3.b Split the set of BFs at the node.

3.c Generate the child nodes of the tree and assign BFs to these child nodes.

The iteration of processing nodes continues until the queue becomes empty as per line 3 in Algorithm 5.2. We next describe each of these sub-steps for a given

Algorithm 5.2: Block tree construction by DO$_i$

```
Input  : BF$_i$ - An inverted index of BFs belonging to DO$_i$
         $b_{min}$ - Minimum block size
         $b_{max}$ - Maximum block size
Output: BT$_i$ - Block tree of database $D_i$
BT$_i$.root $\leftarrow$ makeNode(BF$_i$)                    // Create the root node
$Q \leftarrow$ [BT$_i$.root]                                  // Initialisation of queue
while $Q \neq \emptyset$ do                                   // Iterate until queue is empty
    $v \leftarrow Q.pop()$                                    // Get the current node
                                                              // Sub-step 3.a
    $V_R^i \leftarrow genRatios(v)$                           // Generate local bit ratios
    $V_G \leftarrow secureSummation(V_R^i)$                   // Compute bit ratios globally
    $j_G \leftarrow getBestBit(V_G)$                          // Get the best bit position for splitting
                                                              // Sub-step 3.b
    $v_0, v_1 \leftarrow splitNode(v, j_G)$                   // Split current node data
                                                              // Sub-step 3.c
    if $(|v_0.B| \geq b_{min})$ AND $(|v_1.B| \geq b_{min})$ then   // Check the block sizes
        $v$.left$\leftarrow v_0$                              // Add $v_0$ as the left child of the current node
        $v$.right$\leftarrow v_1$                             // Add $v_1$ as the right child of the current node
        if $(|v_0.B| \geq b_{max})$ then                      // If block too large
            $Q.push(v_0)$                                     // Add to queue for futher processing
        end
        if $(|v_1.B| \geq b_{max})$ then                      // If block too large
            $Q.push(v_1)$                                     // Add to queue for futher processing
        end
    end
end
return BT$_i$
```

iteration in more detail. Figures 5.3 to 5.6 illustrate the steps of this algorithm with an example of three DOs.

3.a **Find best bit position for splitting**

At each iteration the node v that is available at the front of the Q is processed. The function *genRatios()* processes the BFs available in node v to generate a vector V_R^i of length l_{bf}. V_R^i contains the ratios f_{ij} between the number of 0's and 1's for each bit position j in a set of BFs (line 5), as is calculated using Equation 5.1:

$$f_{ij} = abs(0.5 - \frac{o_{ij}}{n_{bf}}), \tag{5.1}$$

where f_{ij} is the 0/1 ratio of bit position j of DO$_i$, o_{ij} is the number of BFs in v that have a 1-bit at position j, $n_{bf} = |v|$ is the size of v, and $V_R^i = [f_{i1}, f_{i2}, f_{i3}, \cdots, f_{il_{bf}}]$.

Database Owner A (DO_A)

	BF$_A$				
bf_A^1	0	1	1	0	0
bf_A^2	1	0	1	0	1
bf_A^3	1	0	0	0	0
bf_A^4	0	1	1	1	0
bf_A^5	0	1	0	0	1
bf_A^6	1	1	0	1	0
bf_A^7	0	1	1	1	0
bf_A^8	0	1	0	0	0

Abs Diff from 0.5

$V_R^A =$	1/8	1/4	0	1/8	1/4

Database Owner B (DO_B)

	BF$_B$				
bf_B^1	1	0	0	1	1
bf_B^2	1	0	1	0	1
bf_B^3	0	0	1	1	0
bf_B^4	1	1	1	1	1
bf_B^5	1	1	0	0	0
bf_B^6	0	0	1	0	1
bf_B^7	0	0	0	1	1
bf_B^8	1	0	1	1	1

Abs Diff from 0.5

$V_R^B =$	1/8	1/4	1/8	1/8	1/4

Database Owner C (DO_C)

	BF$_C$				
bf_C^1	0	1	1	1	0
bf_C^2	0	1	0	1	0
bf_C^3	1	0	1	1	0
bf_C^4	1	1	0	0	1
bf_C^5	0	0	1	0	0
bf_C^6	1	0	0	1	0
bf_C^7	0	1	1	0	1
bf_C^8	0	1	0	1	0

Abs Diff from 0.5

$V_R^C =$	1/8	1/8	0	1/8	1/4

Figure 5.3: Bloom filter generation and calculation of 0/1 bit ratios and absolute differences from 50% filled (line 5 in Algorithm 5.2).

Random vector ($V_{\mathbf{R}}$) = | 10 | 5 | 12 | 13 | 6 |

$V_{\mathbf{R}} + V_R^A$				
10.125	5.25	12.0	13.125	6.25

$\rightarrow$

$V_{\mathbf{R}} + V_R^A + V_R^B$				
10.25	5.5	12.125	13.25	6.5

$\rightarrow$

$V_{\mathbf{R}} + V_R^A + V_R^B + V_R^C$				
10.375	5.625	12.125	13.375	6.75

Globally summed ratio vector (V_G): | 0.375 | 0.625 | 0.125 | 0.375 | 0.75 |

Ranking of bit positions based on V_G: | 2 | 3 | 1 | 2 | 4 |

Best splitting bit position (j_G) = 3

Figure 5.4: Secure summation of absolute differences and selecting best bit position for splitting (lines 6 and 7 in Algorithm 5.2). In this example bit 3 will be selected as the best bit position for splitting.

The bit positions that have a 1-bit in half of the BFs in a given node will have the lowest ratio value of 0, and the bit positions that have a 1-bit or 0-bit in all the BFs are given the highest ratio value of 0.5. An example of this ratio calculation is illustrated in Figure 5.3.

Once all DOs have computed their ratio vectors, V_Rs, locally based on their node data, a common bit position needs to be selected as the best bit for splitting the set of BFs for the child nodes in the next level of the tree. For computing the global bit position j_G, we extend Algorithm 4.1 (on page 70) in the function *secureSummation()* to securely compute the summation of these V_Rs of ratios (each being a vector of length l_{bf}), where each DO's V_R is considered as private input (line 6 in Algorithm 5.2). Similar to Algorithm 4.1, DO_1 generates a random vector $V_{\mathbf{R}}$ of length l_{bf} and follows the steps in Algorithm 4.1 to compute a globally summed ratio vector V_G.

Next the best splitting bit position j_G is calculated and selected by ranking the index positions in V_G according to the summed values. The function *getBestBit()* ranks the index positions in V_G (line 7 in Algorithm 5.2), where the index position with the lowest sum of f_{ij}'s gets the highest rank as

Database Owner A (DO_A)

BF$_A$ 0

bf_A^3	1	0	0	0	0
bf_A^5	0	1	0	0	1
bf_A^6	1	1	0	1	0
bf_A^8	0	1	0	0	0

Abs Diff from 0.5

$V_R^A =$	0	1/4	–	1/4	1/4

Database Owner B (DO_B)

BF$_B$ 0

bf_B^1	1	0	0	1	1
bf_B^5	1	1	0	0	0
bf_B^7	0	0	0	1	1

Abs Diff from 0.5

$V_R^B =$	1/6	1/6	–	1/6	1/6

Database Owner C (DO_C)

BF$_C$ 0

bf_C^2	0	1	0	1	0
bf_C^4	1	1	0	0	1
bf_C^6	1	0	0	1	0
bf_C^8	0	1	0	1	0

Abs Diff from 0.5

$V_C^C =$	0	1/4	–	1/4	1/4

Random vector ($V_\mathbf{R}$) =

12	4	–	15	11

$V_\mathbf{R} + V_R^A$

12.0	4.25	–	15.25	11.25

$\rightarrow$

$V_\mathbf{R} + V_R^A + V_R^B$

12.167	4.417	–	15.417	11.417

$\rightarrow$

$V_\mathbf{R} + V_R^A + V_R^B + V_R^C$

12.167	4.667	–	15.667	11.667

Globally summed ratio vector (V_G):

0.167	0.667	–	0.667	0.667

Ranking of bit positions based on V_G:

1	2	–	2	2

Best splitting bit position (j_G) = 1

Figure 5.5: The Next splitting iteration of the list of BFs in Figure 5.3 where bit position 3 is 0 (i.e. left branch of the block tree). As in Figure 5.3, first the absolute difference of 0/1 bit ratios from 50% filled is calculated for all bit positions except bit position 3. Next a secure summation is performed on these absolute differences and the bit position is selected for splitting. In this example, bit position $j_G = 1$ will be selected as the best position for splitting.

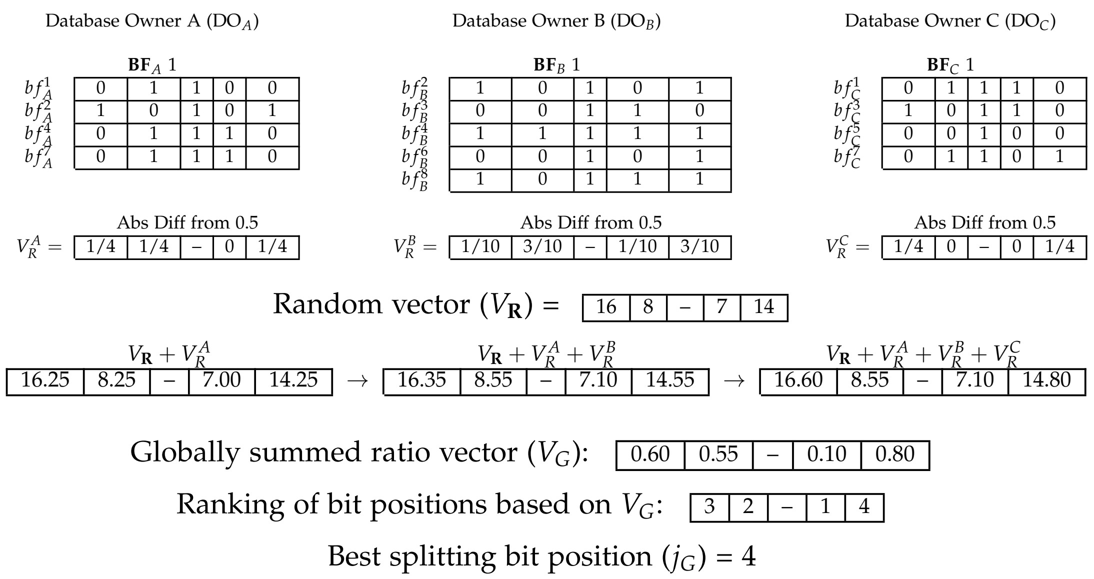
shown in Equation 5.2.

$$j_G = argmin\{j : (\frac{d}{2} - \sum_{j=1}^{d} f_{ij})\}, \tag{5.2}$$

This best index position j_G is selected as the position for splitting the set of BFs in the current node v. Figure 5.4 illustrates an example of computing j_G for the set of BFs illustrated in Figure 5.3. In this example bit position 3 is selected as the best bit position for splitting because it has the lowest sum of ratio values.

3.b **Splitting the set of Bloom filters**

The selected global best bit position j_G is used to split the set of BFs, B, of the current node into two lists of BFs B_0 and B_1. In the function *splitNode()* used in Algorithm 5.2 (line 8), all the BFs that contain a 0 in the best bit position j_G are assigned to the list B_0 (left branch of the tree) and all others are assigned to the list B_1 (right branch of the tree). Next, two new nodes v_0 and v_1 are created, where B_0 and B_1 are assigned as the data of v_0 and v_1, respectively. Figures 5.5 and 5.6 illustrate an example of splitting a set of BFs. The nodes v_0 and v_1 will be added to Q to be processed in the next iterations depending upon the number of BFs contained in B_0 and B_1 as explained next.



Database Owner A (DO$_A$)

BF$_A$ 1

bf_A^1	0	1	1	0	0
bf_A^2	1	0	1	0	1
bf_A^4	0	1	1	1	0
bf_A^7	0	1	1	1	0

Abs Diff from 0.5

$V_R^A =$	1/4	1/4	–	0	1/4

Database Owner B (DO$_B$)

BF$_B$ 1

bf_B^2	1	0	1	0	1
bf_B^3	0	0	1	1	0
bf_B^4	1	1	1	1	1
bf_B^6	0	0	1	0	1
bf_B^8	1	0	1	1	1

Abs Diff from 0.5

$V_R^B =$	1/10	3/10	–	1/10	3/10

Database Owner C (DO$_C$)

BF$_C$ 1

bf_C^1	0	1	1	1	0
bf_C^3	1	0	1	1	0
bf_C^5	0	0	1	0	0
bf_C^7	0	1	1	0	1

Abs Diff from 0.5

$V_C^R =$	1/4	0	–	0	1/4

Random vector ($V_{\mathbf{R}}$) =

16	8	–	7	14

$V_{\mathbf{R}} + V_R^A$

16.25	8.25	–	7.00	14.25

$\rightarrow$ $V_{\mathbf{R}} + V_R^A + V_R^B$

16.35	8.55	–	7.10	14.55

$\rightarrow$ $V_{\mathbf{R}} + V_R^A + V_R^B + V_R^C$

16.60	8.55	–	7.10	14.80

Globally summed ratio vector (V_G):

0.60	0.55	–	0.10	0.80

Ranking of bit positions based on V_G:

3	2	–	1	4

Best splitting bit position (j_G) = 4

Figure 5.6: Second splitting iteration of list of BFs in Figure 5.3 where bit position 3 is 1 (i.e. right branch of the block tree). As in Figure 5.3, first the absolute difference of 0/1 bit ratios from 50% filled is calculated for all bit positions (except bit position 3). Next a secure summation is performed on these absolute differences and bit position $j_G = 4$ (with the lowest ratio value) is selected for splitting.

3.c **Generate the child nodes**

According to the selected best bit position, j_G, the two lists B_0 and B_1 may contain an uneven number of BFs, i.e. one list contains a smaller number of BFs than the other. The disadvantage of such a division is that sensitive information can potentially be revealed about the blocks that are having a smaller number of BFs, where an adversary can potentially re-identify individual records [200].

As a solution the proposed parameter *minimum block size* (b_{min}) guarantees that every block in a given block tree structure contains at least b_{min} records. After splitting, if any of the resulting two lists, B_0 and B_1, of a given node contains less than b_{min} records, then these lists will not be assigned to any new node in the block tree. Instead, the original set of BFs in that node, B, is included as a relevant block in the parent of the current node.

If the two lists, B_0 and B_1, contain a number of records greater than b_{min}, then these newly created nodes are assigned as child nodes to the current node, i.e. the node with the list B_0 becomes the left child of the current node and the list B_1 becomes the right child respectively (lines 10 and 11 in Algorithm 5.2).

One important consideration in the tree construction phase is to have control over the number of blocks created in the tree. The parameter *maximum*

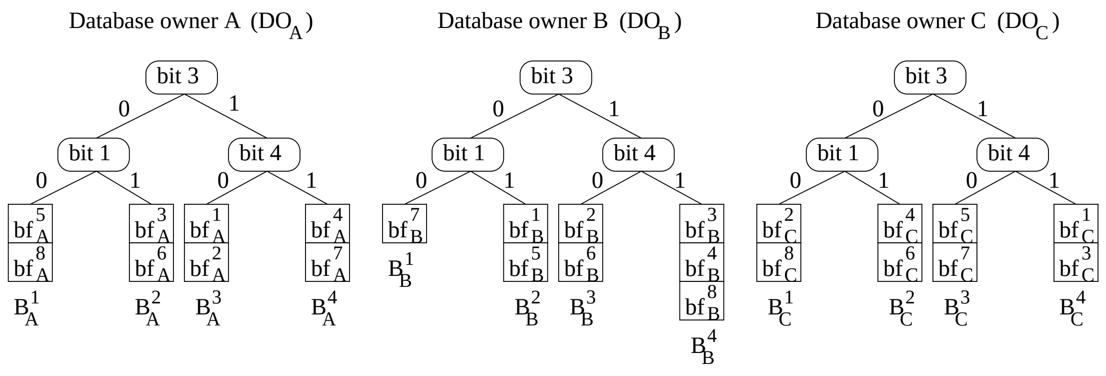


Figure 5.7: The resulting three single-bit trees as generated by the database owners DO_A, DO_B, and DO_C based on the example in Figures 5.3 to 5.6 with blocks across the three databases.

block size (b_{max}) allows the user to control the maximum number of records that can be contained in a block, which indirectly controls the number of iterations that occur when creating sub-trees. After splitting the current node's set of BFs, the two resulting lists, B_0 and B_1, are checked against b_{max} (lines 12 and 15).

If the number of records in both B_0 and B_1 is greater than b_{max}, then these two child nodes are added to the queue Q for future splitting (lines 13 and 16). Therefore the splitting process continues until all generated blocks are within the size range of $[b_{min}, b_{max}]$. Figure 5.7 illustrates the resulting block tree structures for the set of BFs from Figure 5.3 after three splitting iterations were conducted, where bit position 3, 1, and 4 were selected, respectively, in each iteration.

5.4 Conceptual Analysis of Tree based Blocking

In this section we analyse our single-bit tree based blocking approach in terms of complexity, quality of blocking, and privacy.

5.4.1 Complexity

We analyse the computational and communication complexities of our blocking approach in terms of a single database owner (DO). Let us assume there are $n_r = |\mathbf{D}|$ records in a database with each having an average of n_q q-grams. In the record encoding step all the records are encoded using n_h hash functions. Therefore, the second step of our approach has a computational complexity of $O(n_h \cdot n_q \cdot n_r)$ for the encoding of all records in $\mathbf{D}$ into BFs.

In the third step the block tree construction starts once the records are encoded into n_r BFs. As we discussed in the previous section, the parameter b_{max} is used to control the number of blocks, which indirectly controls the number of levels generated in the trees. At one extreme, if b_{max} is equal to n_r, then the number of levels in the trees becomes 1 (all the records are assigned to the root node), while at the other extreme if b_{max} is equal to $n_r/2$ then the trees will have two levels (root node with two child nodes). When b_{max} is equal to 1 the single-bit tree is constructed with $log_2(n_r)$ levels. Therefore the number of levels in a single-bit tree can be calculated as $log_2(\frac{n_r}{b_{max}})$.

At each level of the block tree a total of n_r records are processed, where combined all child nodes at a given level hold a total of n_r records. Therefore the insertion of n_r BFs into a block tree requires a computational complexity of $O(n_r \cdot log(\frac{n_r}{b_{max}}))$. Also, for a given iteration in the block tree construction, the computation of the secure summation protocol requires each DO to process the set of BFs that is assigned to the currently processed node, which requires each BF to be scanned for each bit position to get a count of the number of 1's. Therefore, line 5 of Algorithm 5.2 has a computational complexity of $O(l_{bf} \cdot n_r)$ for each level in the tree.

In our approach, the DOs only need to communicate with each other in the parameter agreement step and to perform the secure summation protocol to find the best bit for splitting. This requires a communication between the DOs when each node is created. By assuming each DO is directly connected to all other DOs, the distribution of the final ratio vector V_G to d DOs requires d messages for each node in the block tree structure, where each message is of size of l_{bf} bits, where l_{bf} is the length of a BF. Therefore the entire approach has a communication complexity of $O(l_{bf} \cdot d \cdot log_2(\frac{n_r}{b_{max}}))$ for d DOs.

5.4.2 Blocking Quality

We analyse the quality of our approach in terms of effectiveness, which requires that all similar records to be grouped into the same block, and efficiency, which requires the number of candidate record tuples (CRT), as described on page 65, generated to be as small as possible while including all true matching record sets [216]. By assuming each block tree of d DOs contains $n_B = \frac{n_r}{b_{max}}$ blocks, each containing b_{max} records, the number of CRTs generated by our approach is equal to $n_B \cdot b_{max}^d$ for d databases.

The parameter b_{max} decides the number of child nodes that are created in a block tree, which in turn controls the number of blocks generated. If the value of b_{max} is increased, then the number of blocks that are generated becomes lower. However, a larger b_{max} value increases the number of CRT comparisons required by the comparison and classification step which would increase the overall runtime of the linkage process. Therefore, b_{max} needs to be selected by considering factors such as the database sizes (n_r) and the number of DOs (d) such that both effectiveness and efficiency are achieved while guaranteeing sufficient privacy as well.

5.4.3 Privacy

We assume all DOs follow the honest-but-curious (semi-honest) adversary model [3, 186], where each DO follows the steps of the approach while trying to find as much as possible about the databases being blocked by the other DOs. We consider the amount of information a DO can learn about other databases in the tree construction step in this privacy analysis. In our approach, as described above, the DOs communicate with each other to compute the global best bit position for splitting.

During the secure summation, each DO sums its ratio vector V_R with the partial resulting vector sent to it by the previous DO, but a DO will not be able to learn any information about the ratio values of another database since the random vector $V_{\mathbf{R}}$ is only known to the DO that initiated the protocol. Once the first DO received the final partial sum vector, it subtracts the random vector $V_{\mathbf{R}}$ from the summed values, but is not capable of learning anything about the other ratio vector values.

However, as we have discussed in Section 2.4 on page 33, collusion is possible between DOs under the honest-but-curious model. Collusion between several DOs compromises the privacy of the secure summation protocol. Therefore, as we have detailed in Section 4.3 on page 70, a more collusion resistant secure summation protocol can be used to overcome such attacks.

At the end of our blocking approach, a private comparison and classification technique [9, 59, 208, 212] can be used to compare each respective set of blocks among the DOs, which should not reveal any information regarding the sensitive attribute values and non-matching record sets. However, the privacy of such a technique is outside the scope of this chapter.

Also, our approach allows each DO to perform a generalisation strategy on its blocks that makes the re-identification from the BFs not possible [200]. The parameter *minimum block size* (b_{min}) is used to guarantee that every block in the block tree structure contains at least b_{min} BFs. This ensures all blocks that are generated have the same minimum number of BFs, which makes a frequency attack or a dictionary attack, where an adversary hash-encodes values from a large publicly available database using existing hash encoding functions, much more difficult [212].

However, due to the iterative splitting process in the block tree construction, a DO can learn partial information about the BFs of other DOs based on the bit positions selected for splitting. For example, in a given splitting iteration the list of BFs in the left most node in a block tree contains 0's and the BFs in the right most node contains 1's for all previously selected splitting bit positions. An attacker could use this information to conduct a cryptanalysis on the BFs to identify q-grams encoded in these BFs [39, 132]. Developing such a cryptanalysis method for a blocking tree structure to identify the attribute values that are encoded in BFs for privacy evaluation is left for future work.

Table 5.2: Parameter settings used in the experimental evaluation.

Parameter	Value range
Dataset name	NC-SYN
Number of databases (d)	3, 5, 7, 10
Number of records in each database (n_R)	5,000 to 1,000,000
Blocking key attributes (A)	Given name, Surname, City (suburb), and Postcode
Corruption levels	0%, 20%, and 40%
Bloom filter length (l_{bf})	500, 1000, 2000
Number of hash functions (n_h)	30 and optimal (calculated as discussed in Section 4.3 on page 68)
Character length of q-gram (n_q)	2, 3, 4
Minimum block size (b_{min})	25, 50, 250, 500
Maximum block size (b_{max})	50, 100, 500, 1000

Table 5.3: Average maximum memory in Megabytes (MB) used by the tree based blocking approach.

Database size (n_r)	BF generation	BT construction
5,000	20	23
10,000	33	47
50,000	112	117
100,000	340	370
500,000	970	987
1,000,000	$1,920$	$1,958$

5.5 Experimental Evaluation and Discussion

In this section, we present and discuss the results of the experimental evaluation of our tree based blocking approach conducted on the datasets described in Section 4.4 (on page 78). Table 5.2 summarises the parameter values used in our approach.

5.5.1 Scalability

Figures 5.8 shows the scalability of our approach in terms of computation and communication costs. We measured the average runtime required for generating Bloom filters (BFs) and for constructing a block tree (BT) for a single database. As shown in Figure 5.8 (a), the average runtime for the BT construction increases linearly with the database size. As we expected the BF generation increases linearly with database size as more time is required to encode more records into BFs.

As shown in Figure 5.8 (b), the total runtime required for our approach increases with the number of databases to be blocked as more communication is required in the BT construction process. Figure 5.8 (c) shows that more messages are communicated as the number of database owners (DOs) increases, which increases the overall communication costs. As the database sizes increase, more messages need to be sent between DOs since more splitting iterations are required in the BT construction step.

We also measured the average maximum memory consumption per DO for the two steps BF generation and BT construction of our approach. As shown in Table 5.3, the average memory required increases with the database size, however, it consumed less than 2 GByte of memory for blocking a database with 1 million records.

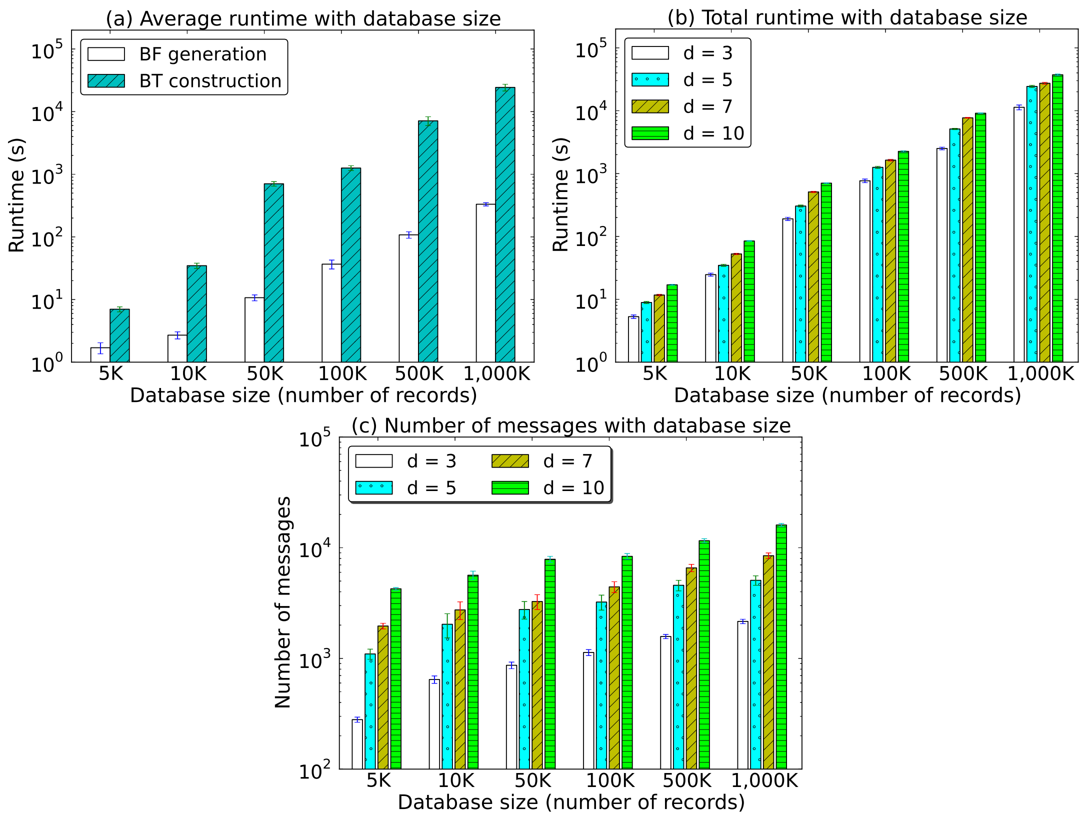


Figure 5.8: (a) Average runtime for Bloom filter (BF) generation and block tree (BT) construction, (b) total runtime with different number of databases to be blocked, and (c) total number of messages communicated in the BT construction with different database sizes. K represents 1000 records. Note that plots have different y-axis scales.

As illustrated in Figures 5.9, we also investigated the runtime of BF generation with different number of hash functions (n_h), q-gram lengths (l_q), BF length (l_{bf}), and blocking key attributes (A). As we described in Section 4.3 (on page 68), the BF generation step required longer runtime with the optimal number of hash functions used when the BF length increases, as shown in Figure 5.9 (a). This occurs because more hash functions are required to encode each q-gram into a BF when l_{bf} is increasing under the optimal BF parameter settings.

As can be seen in Figure 5.9 (b), the runtime required to encode an attribute value into a BF decreases with l_q as less q-grams are generated for a record. However, as shown in Figure 5.9 (b), more runtime is required with BFs with a larger l_{bf} compared to a smaller length even for a larger q-gram length since more hash functions are used to encode an attribute value. As can be seen in Figure 5.9 (c), the BF generation time of our approach increases with the number of blocking key attributes because more q-grams are generated for a record.

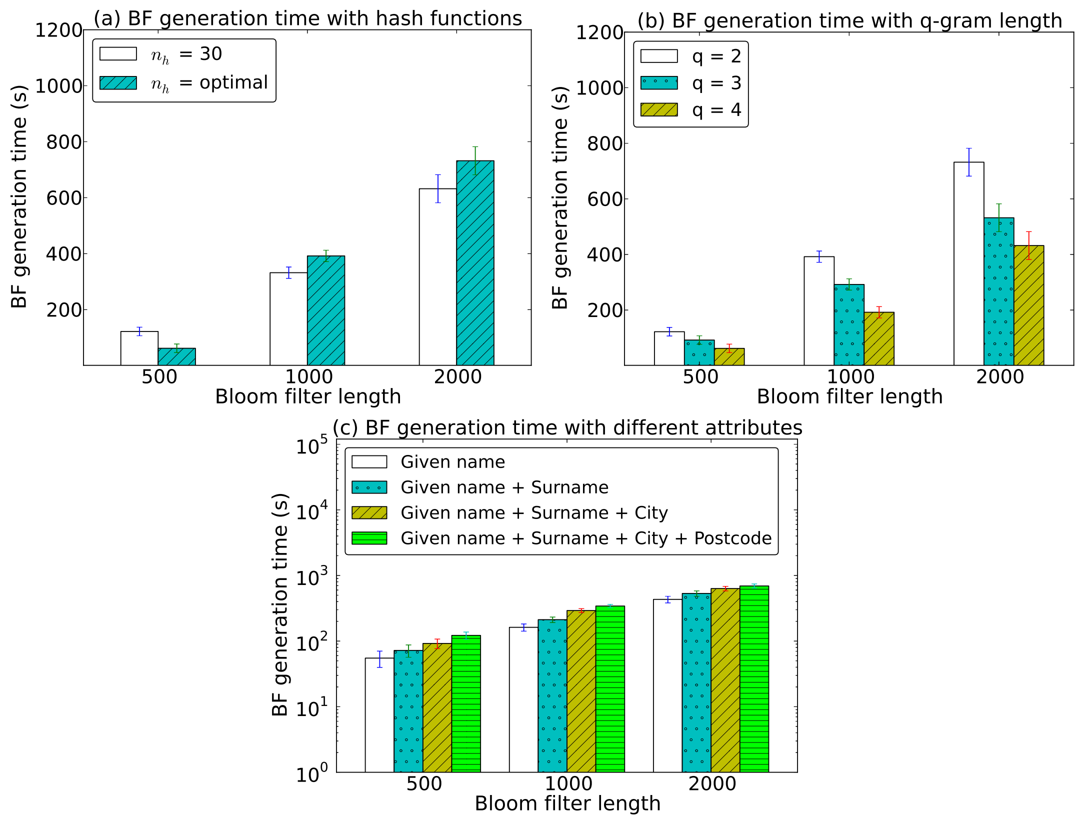


Figure 5.9: Average BF generation time for different (a) number of hash functions (n_h), (b) q-gram length (l_q), and (c) blocking key (A) attributes. Note that plots have different y-axis scales.

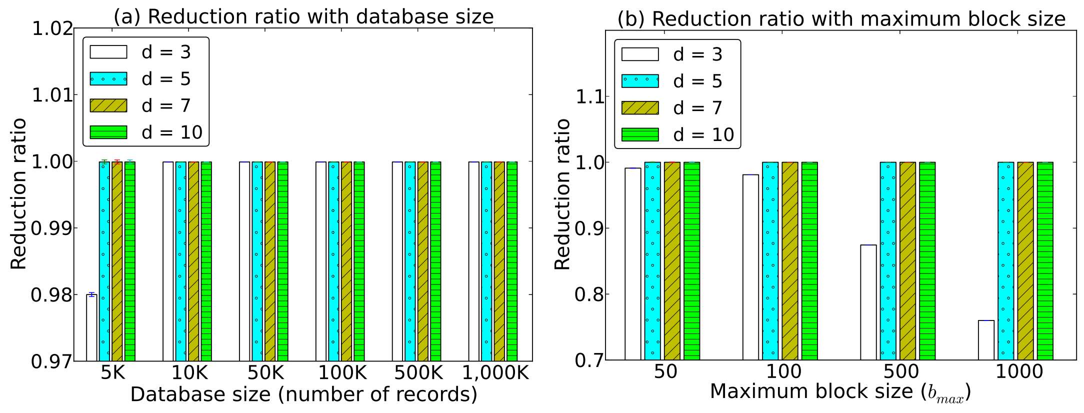


Figure 5.10: Reduction ratio (RR) with different (a) database sizes and (b) maximum block sizes. Note that plots have different y-axis scales.

As we have described in Section 2.5 on page 35, the scalability of a blocking technique can also be measured based on the number of record comparisons. We used the adopted reduction ratio (RR) metric, which we described in Section 4.4 (see page 82)

Table 5.4: Number of candidate record pairs (*CRT*s) generated for the linkage of 3 databases.

Database size (n_r)	without blocking	with our approach ($b_{max} = 50$)
5,000	1.25 x 10^{11}	1.64 x 10^{9}
10,000	1.0 x 10^{12}	1.73 x 10^{9}
50,000	1.25 x 10^{14}	3.78 x 10^{11}
100,000	1.0 x 10^{15}	1.66 x 10^{12}
500,000	1.25 x 10^{17}	3.14 x 10^{12}
1,000,000	1.0 x 10^{18}	6.67 x 10^{14}

to measure the scalability of our approach. As can be seen in Figure 5.10 (a), RR remains closer to 1 for different database sizes and for a different number of databases (when d = 5, RR $\approx$ 0.9999), which illustrates our approach is reducing the number of total candidate record tuples (*CRT*s) that need to be compared dramatically. The reason for this behaviour is that once the blocks are generated only the blocks of each corresponding leaf in the block trees are compared across all the databases to be blocked. Table 5.4 shows the number of candidate record pairs (*CRT*s) generated for the linkage of 3 databases with our blocking approach and without any blocking.

We also investigated the RR with different maximum blocks sizes (b_{max}). As shown in Figure 5.10 (b), RR decreases with the increase of b_{max}. This is because more *CRT*s are generated for larger blocks compared to blocks with a smaller number of records. A larger b_{max} also reduces the overall runtime of our approach as less splitting iterations are required in the BT construction, which reduces the total number of messages to be communicated between DOs. However, we have seen that for a large number of databases RR remains closer to 1 even for larger b_{max} since the number of *CRT*s generated by our approach is significantly smaller compared to the total record comparisons required between databases (see Figure 5.10 (b)).

5.5.2 Blocking Quality

We measure the blocking quality of our approach in terms of the pairs completeness (PC) and F-measure (FM), as we described in Section 4.4 on page 83. We used the NC-SYN datasets with corrupted records (as detailed in Section 4.4 on page 78) to evaluate the blocking quality of our approach. We used the datasets with corruption levels 0%, 20%, and 40% as these datasets represent realistic scenarios with *dirty* data [40].

As illustrated in Figure 5.11 (a), our blocking approach achieves a PC of 1.0 for 0% corruption with different number of databases d. However, we noted that the blocking quality of our approach is affected by the quality of the data and PC is decreasing rapidly with the number of databases with low quality data (20% and 40% corruption levels). Since BFs of the corrupted records of the same entity result in different bit patterns (due to the different q-grams extracted from corrupted attribute values), these BFs will be assigned to different blocks in the splitting process. This results in similar records not being compared in the comparison step which

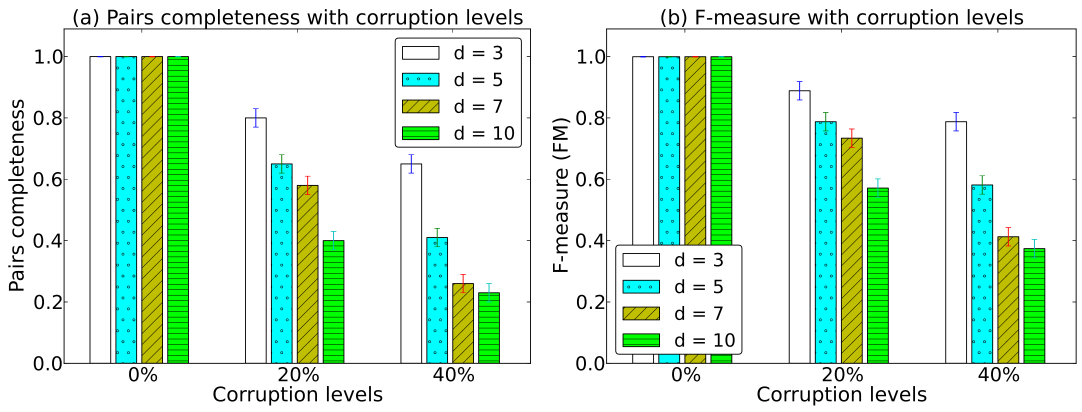


Figure 5.11: (a) Pair completeness (PC) and (b) F-measure (FM) with different number of databases for different corruption levels.

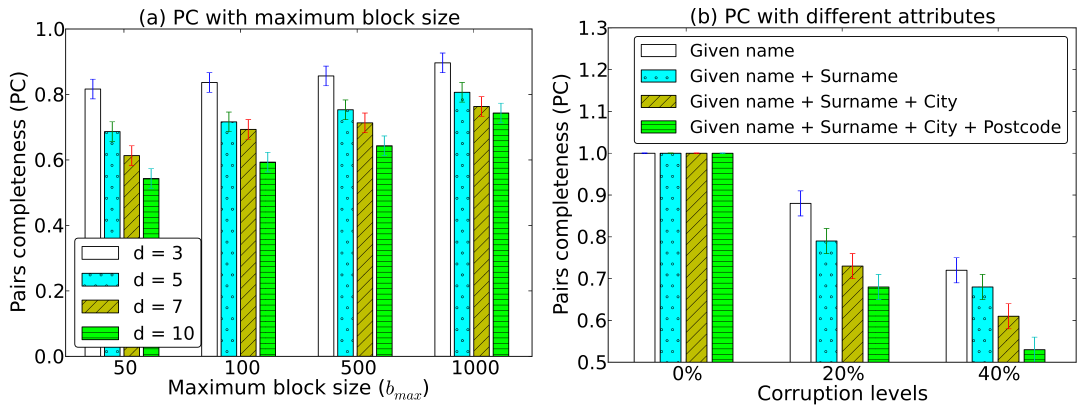


Figure 5.12: Pair completeness (PC) with different (a) maximum block sizes and (b) blocking key attributes for different corruption levels. Note that plots have different y-axis scales.

potentially reduces the overall quality of the linkage. Figure 5.11 (b) shows that FM also decreases with the corruption level as the number of databases to be blocked increases.

As shown in Figure 5.12 (a), PC increases when the size of the blocks is increased. The block sizes are controlled with the parameter b_{max}, where high PC values are achieved with larger b_{max} values. This is because more *CRT*s are generated for larger b_{max} values which potentially increases the number of true matches in blocks by assigning similar records into the same block. Figure 5.12 (b) illustrates PC with different corruption levels for different blocking key attributes. It shows that for databases with corrupted records high PC values can be achieved by using less blocking key attributes compared to using more attributes. Therefore, in applications where databases are having more corrupted records the use of few attributes for blocking could decrease the overall number of missed true matches in our approach.

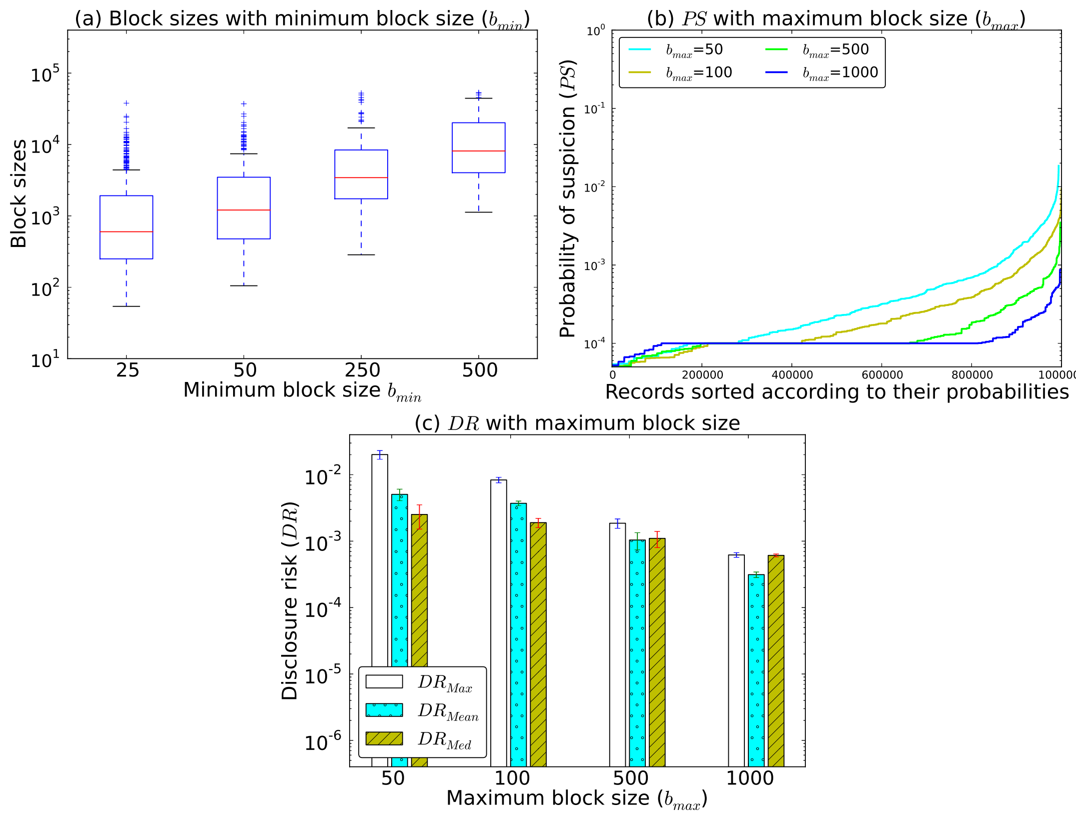


Figure 5.13: (a) Average block sizes with minimum block size (b_{min}), (b) probability of suspicion (PS), and (c) disclosure risk values with maximum block size (b_{max}) for a 1,000K database. K represents 1,000 records.

5.5.3 Privacy

To evaluate the privacy of our approach we studied the block sizes generated for different minimum block size (b_{min}) values (with $b_{max} = 2 \cdot b_{min}$), as shown in Figure 5.13 (a). This figure illustrates that our approach guarantees that all generated blocks contain at least b_{min} records ensuring k-anonymous privacy where k = b_{min}. However, we noted that for smaller b_{min} values more large size blocks are generated due to the iterative splitting step. This could increase the overall number of CRTs to be compared in the comparison step of the PPRL process.

We computed average probability of suspicion (PS) values (for the frequency based linkage attack described in Section 2.4 on page 38) for a database with 1 million records for different b_{max} by assuming all block trees contain similar block structures. As shown in Figure 5.13 (b), our approach is having a maximum PS value of less than $(1/b_{min})$ for each b_{max} value, which indicates a record in a block can be matched to more than b_{min} values in a global database $\mathbb{G}$ (under the worst case assumption of $\mathbb{G}$ being equal to the blocked database). For b_{max} = 50 our blocking approach has a maximum PS of 0.0192 because the smallest block contains 52 records (see Figure 5.13 (b)).

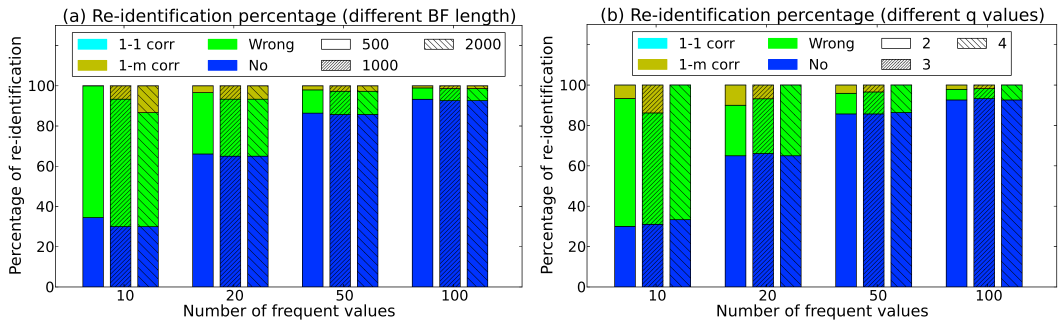


Figure 5.14: A cryptanalysis attack performed upon the generated blocks of a 1,000K database with different frequent values for (a) different Bloom filter length and (b) different q-gram length.

We also measured the average (DS_{Mean}), median (DS_{Med}), and maximum (DS_{Max}) disclosure risk values with different b_{max} values (as we described in Section 4.4). As shown in Figure 5.13 (c), the DR values decrease for large block sizes as more records are included in a given block making our blocking approach more secure against frequency attacks.

Finally, we conducted the cryptanalysis attack we described in Section 4.5 on page 85. We conducted this attack assuming an external attacker under the scenarios of an attacker gains access to a generated block of a database. In here an attacker tries to re-identify an attribute value encoded in a BF by using the frequency distributions of 1-bits in the BFs and q-grams of a publicly available $\mathbb{G}$. However, note that such attacks are realistically highly unlikely since only the database owners (DOs) are participating in the block generation process and each DO does not send its own database or a block to any other party during the block tree construction process.

We evaluate the accuracy of our attack by calculating (1) the percentage of correct guesses with 1-to-1 matching, (2) the percentage of correct guesses with 1-to-m (many) matching, (3) the percentage of wrong guesses, and (4) the percentage of no guesses, where these four percentages sum to 100. These four categories are labelled as *1-1 corr*, *1-m corr*, *Wrong*, and *No* in Figure 5.14, respectively.

For the first scenario, we conducted the attack for each generated block of a database with 1 million records. We set the parameter b_{max} and blocking key attribute (A) to $1,000$ and *Given name*, respectively, and performed the cryptanalysis for different BF lengths and q-gram values. As illustrated in Figure 5.14, an attacker could not re-identify a value of A as block contains no enough frequency information to identify q-grams that are encoded in the BFs. Therefore, conducting such cryptanalysis to correctly re-identify attribute values at the block level is impossible.

5.6 Summary

In this chapter, we have presented a novel blocking approach for MD-PPRL based on Bloom filters and a single-bit tree data structure. Each database owner constructs the block tree structure based on the Bloom filters generated on their database. The database owners communicate with each other using a secure summation protocol to collaborate on the construction of their blocking tree structures. The proposed approach was validated by an experimental evaluation, where we performed experiments on different databases of size of up-to one million records.

The evaluation results indicated that our approach scales linearly with both the size of the databases to be linked and the number of database owners. As described in Section 5.4.2, the selection of minimum (b_{min}) and maximum (b_{max}) block sizes controls the runtime and blocking quality of this approach. A specific future research avenue is to tackle the problem of finding the optimal value for b_{min} and b_{max} by considering the scalability and blocking quality. Another extension of our current work is to investigate the parallelisation of Algorithm 5.2, which can further improve the performance of our approach. Extending Algorithm 5.2 to different adversary models is another future research avenue of our approach.

As we have seen in Section 5.5, due to the recursive splitting process in the block tree construction, similar Bloom filters could be assigned to different blocks. This requires an additional step in the block generation process to reassign these similar Bloom filters again into the same block. In the next chapter, we aim to study how a clustering approach can be used to minimise such wrong assignments of Bloom filters into different blocks. We also investigate how to improve the runtime of our current work by selecting multiple bits for splitting in each iteration.

Chapter 6

Clustering based Blocking for Multidatabase Privacy-Preserving Record Linkage

In this chapter we propose a clustering based scalable blocking approach for privacy-preserving record linkage (PPRL) between multiple (more than two) databases where database owners collaboratively participate in the block generation process. In Sections 6.1 and 6.2 we provide an introduction and overview of our blocking approach, respectively. As we describe in Section 6.3, we propose a clustering based splitting and merging technique to generate blocks of the databases. In Section 6.4, we provide a detailed analysis of our approach with regard to complexity, blocking quality, and privacy. In Section 6.5 we then validate this analysis through an experimental study, and finally we summarise our findings in Section 6.6. Table 6.1 summarises the notation we use in this chapter.

6.1 Introduction

As detailed in Chapter 2, a blocking technique should output a reduced set of candidate record pair or sets for comparison and classification by keeping true matching record pairs or tuples while removing as many of the true non-matching record pairs or tuples as possible. As a result, expensive similarity comparisons are then only required on a smaller number of candidate record pairs or tuples. In our previous tree based blocking approach described in Chapter 5, we noted that the blocks generated across different databases could result in low blocking quality because of the recursive splitting of the binary tree data structures which can lead to a reduction of the overall quality and accuracy of the linkage.

With the primary aim of improving the quality in blocking, in this chapter we propose a novel blocking approach that uses a splitting and merging technique to generate blocks. In this approach we first split the databases into smaller blocks, which we call *mini-blocks*, and then we merge these mini-blocks based on their similarity by using a clustering algorithm.

Table 6.1: Notation and terminology used in this chapter

B	Set of blocks
BF	Inverted index of Bloom filters
D	A database
MB	Set of mini-blocks
A	Set of blocking attributes
B	A block
BC	Set of bit combinations
I	List of splitting bit positions
MB	List of mini-blocks of a given block B
Q	A queue data structure
$R, R.a, R.id$	A record, record attribute value, and record identifier
S	A set of q-grams
V_C, V_R, V_G	Combination vector, ratio vector, and a global summed ratio vector
bf	A Bloom filter
b_{min}	Minimum merged block size
bs_t	Bit selection threshold
c, c_j	A medoid
d	Number of database owners (or databases)
f_{ij}	Ratio of bit position j of DO$_i$
l_q, l_{bf}	Length of a q-gram and a Bloom filter
mb	A mini-block
mb_{min}, mb_{max}	Minimum and maximum size of a mini-block
n_b, n_h, n_q	Number of splitting bit positions, hash functions and q-grams
o_{ij}	Number of 1's in bit position j for DO$_i$
$sim_H()$	A Hamming distance based similarity function
ts_t, ls_t	Tight and loose similarity thresholds
BF	Bloom filter
BK	Blocking key attribute
DO	Database owner
PPRL	Privacy-preserving record linkage
MD-PPRL	Multidatabase PPRL
MDRL	Multidatabase record linkage

Clustering is the process of grouping records or data objects such that records within a cluster are similar to each other, while dissimilar records are in different clusters. As discussed in Section 3.1, few clustering approaches have been adapted for private blocking [110, 134, 215]. Karakasidis et al. [110] proposed a blocking mechanism based on k-nearest neighbour clustering that uses reference values to generate clusters. Following the same concept of using reference values, Vatsalan et al. [215] suggested an two-party private blocking technique based on a sorted nearest neighbourhood clustering approach which provides k-anonymous privacy. In their approach, linkage quality and privacy mainly depend on the reference values selected for blocking. A private blocking mechanism based on hierarchical clustering was introduced by Kuzu et al. [134], however this approach is not scalable due to the computationally expensive similarity calculations it requires. Furthermore, neither of these techniques considered blocking of more than two databases.

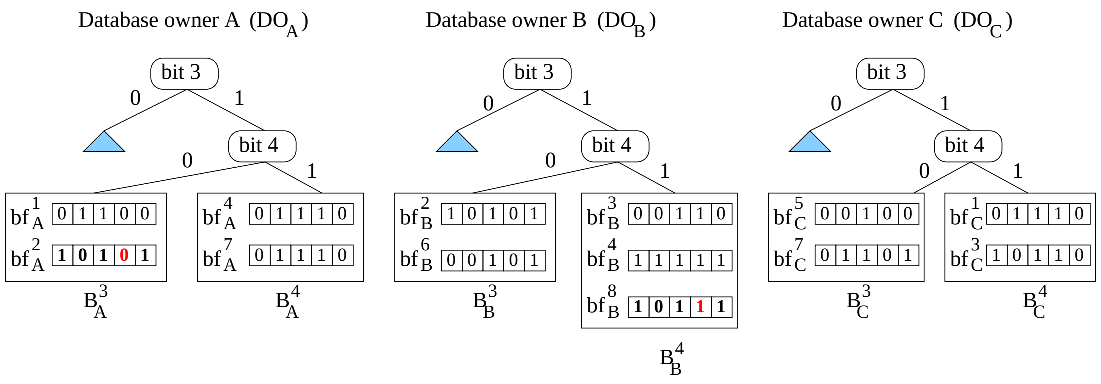


Figure 6.1: An example splitting iteration that illustrates how similar Bloom filters (BFs) are assigned to different blocks due to bit selection in the splitting process (from Figure 5.7). In this example BFs bf_A^2 and bf_B^8 (shown in bold) are assigned to different blocks (B_A^3 and B_B^4, respectively) since they have different values in bit position 4 (shown in red), although they have the same bit values in all other positions. This leads to bf_A^2 and bf_B^8 potentially not being compared in the comparison step, ultimately resulting in a missed match (a *false negative*). The blue triangles represent sub-trees that are not shown.

In our approach we use canopy clustering [48, 144] for merging mini-blocks, which is a technique for clustering large high dimensional databases that has not been applied in PPRL so far. Canopy clustering can achieve a computationally cheap generation of candidate record tuples (*CRT*s), as defined on page 65, by efficiently calculating distances between blocking key values. Records are inserted into one or more overlapping clusters based on their similarity to the cluster medoids (a record that has the highest average similarity to all the other records in a cluster). Each cluster then becomes a block from which *CRT*s are generated.

6.2 Overview of our approach

As detailed in Chapter 2, a blocking technique used in any linkage application should be capable of reducing the comparison space as much as possible without removing record comparisons that correspond to true matches. Using an iterative splitting method, our tree based blocking approach described in Chapter 5 groups records into different blocks depending upon the 1/0 ratio values held at each bit position in the Bloom filters (BFs) selected for splitting. However, at a given iteration two similar records might be blocked into two different blocks even if they only differ by a single splitting bit position selected at the given iteration. This can cause our tree based blocking approach to miss true matches when constructing the block tree structures since the same corresponding blocks in each tree structure are compared with each other, which can result in a decrease of the quality of the overall linkage process. An example of such a splitting iteration is shown in Figure 6.1.

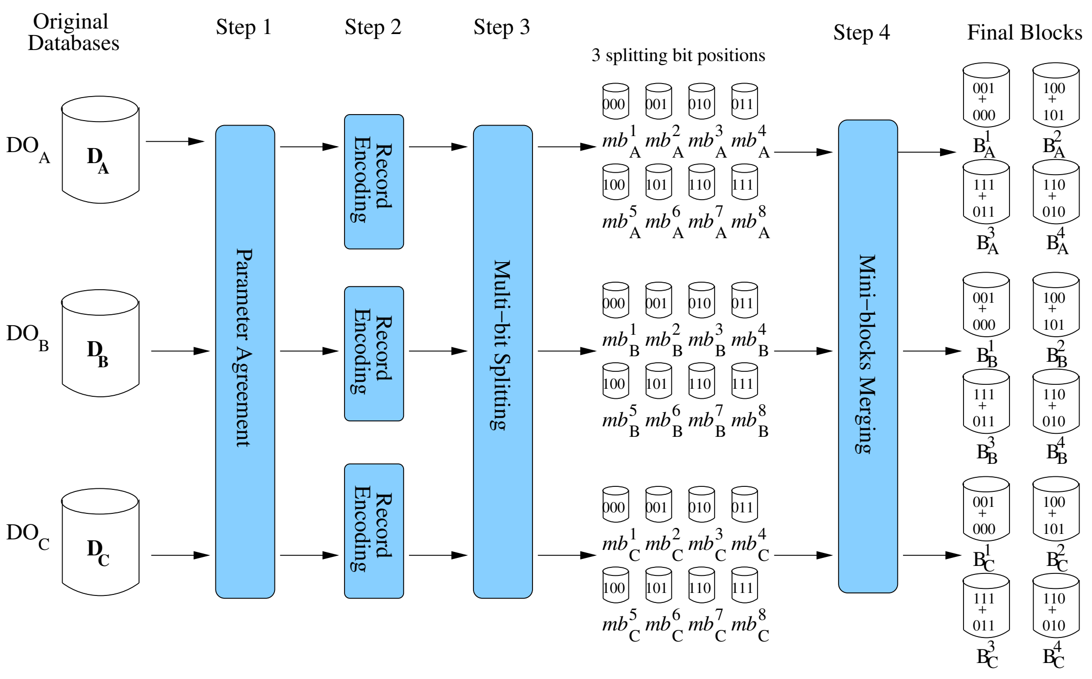


Figure 6.2: A high-level overview of our clustering based blocking approach. In this example $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}_C$ represent the databases held by database owners DO$_A$, DO$_B$, and DO$_C$, respectively. Once the blocks are generated in step 4, the corresponding blocks from different databases can be compared using a private comparison and classification technique [216]. For example, block tuples $\langle B^1_A, B^1_B, B^1_C\rangle$, $\langle B^2_A, B^2_B, B^2_C\rangle$, $\cdots$, and $\langle B^4_A, B^4_B, B^4_C\rangle$ are compared.

To address this issue, we propose a novel blocking approach which follows a split and merge technique to generate blocks. The aim of our approach is to split the databases iteratively into smaller mini-blocks and then merge them together until each final block contains a specific number of records. The merging process is conducted based on the similarities calculated between the medoids of these mini-blocks. The goal of this merging process is to minimise the quality loss occurred in the splitting process by grouping similar records into the same block that were assigned to different blocks during the splitting process. A high-level overview of our approach is shown in Figure 6.2.

As illustrated in Figure 6.2, our approach has four main steps. In the first step the database owners (DOs) need to agree upon the parameter settings used in our approach. The set of parameters used will be explained in more detail in Section 6.3. Once all the DOs have agreed upon the set of parameters, each record in their databases needs to be encoded before these records are grouped into blocks. Similar to our previous tree based blocking approach, Bloom filters (BFs) are used to encode records as BF encoding is the only applicable technique that can be used in the second step of our approach.

As shown in Figure 6.2 in the third step of our approach the DOs split the generated BFs iteratively into a set of smaller blocks which we call *mini-blocks*. In contrast to our previous approach, the proposed approach selects multiple bit positions to split the BFs in a given iteration in order to make the splitting process more efficient. However, all the DOs collaboratively participate in computing the bit positions for splitting in each iterations. The details of the splitting step will be described in Section 6.3. In the example given in Figure 6.2, three bit positions have been selected for splitting, resulting in eight mini-blocks in step 3.

In the fourth step of our approach, the DOs merge the generated mini-blocks until the size of a merged block reaches an acceptable minimum size. The DOs communicate among themselves to decide which similar mini-blocks to merge. As shown in step 4 in Figure 6.2, mini-blocks with bit patterns (000, 001), (111, 011), (101, 100), and (110, 010) are merged together. Finally these merged blocks can be compared using private comparison and classification techniques to determine the matching record sets in different databases [210].

We use the q-grams, BFs, and our extended secure summation protocol as described in Chapter 4 as building blocks of the proposed approach and we refer the reader to Section 4.3 for more details of each of these building blocks. We next describe each step of our approach in more detail.

6.3 Clustering based Scalable Blocking Approach

We assume d DOs are participating in this blocking approach each with a database, and $d \geq 3$. We assume a set A of *quasi-identifiers* [94] (QID), such as first name, last name, address details, etc., are common to all the databases. These attributes in A are used as blocking key (BK) attributes to group similar records into blocks.

1. **Parameter Agreement**

   In step 1 all DOs need to agree upon the set of parameter values to be used. For the record encoding process in step 2, the DOs needs to agree upon the length of a BF, l_{bf}, the length (in characters) of grams, l_q, the n_h hash functions, and the set A of BKs.

   To control the size of the mini-blocks generated in multi-bit splitting in the third step of our approach, all DOs need to agree upon the minimum and the maximum size of a mini-block, mb_{min} and mb_{max}, respectively. Also, all the DOs need to agree upon the maximum number of bit positions, n_b, and the bit section threshold, bs_t, which will be used to find the best splitting bit positions in each iteration, as described below.

   For the merging of mini-blocks in the forth step of our approach, the DOs need to agree upon two thresholds of the clustering technique which are a tight similarity threshold, ts_t, and a loose similarity threshold ,ls_t, and the minimum size (number of records) of a merged block, b_{min}. We will describe the use of each of these parameters in more detail in the following sections.

2. **Record Encoding**

   In the second step of our approach, each DO encodes the records in its database **D** into BFs. To generate BFs for each **D**, we use Algorithm 5.1 in Section 5.3 (on page 93), where each DO iterates over its database and each record R is encoded in a BF bf. Each bf is then added to an inverted index data structure, **BF**, using its record identifier (*R.id*) as its key. **BF** is used in the next step of our approach to construct the set of mini-blocks.

3. **Multi-bit Splitting**

   As the third step of our approach, the generated set of BFs need to be split into sets of mini-blocks. The parameters mb_{min} and mb_{max} specify the lower and upper bounds on the size of a mini-block, respectively. The overall splitting approach for a given database owner, DO$_i$, with $1 \geq i \geq d$, is outlined in Algorithm 6.1.

   In line 1, each DO adds its **BF** into a queue Q as a single block. In each iteration the first block of BFs, B, that is available at the beginning of Q is processed (line 3). In line 4, the function *genRatios()* calculates a vector V_R of length l_{bf} that contains the ratios between the number of 0's and 1's for each bit position of the BFs in B, using Equation 6.1:

$$f_{ij} = abs(0.5 - \frac{o_{ij}}{n_{bf}}), \tag{6.1}$$

   where f_{ij} is the 1/0 ratio of bit position j of DO$_i$, o_{ij} is the number of 1's in position j, and n_{bf} is the number of BFs available in B, i.e. $n_{bf} = |B|$. The lowest ratio value is given to the bit positions that have a value of 1 in half of the BFs and the highest ratio value of 0.5 is given to the bit positions that have a value of 1 or 0 in all the BFs in B.

   Once all DOs have calculated their ratio vectors locally the function *secure-Summation()* performs an extended secure summation protocol, as described in Chapter 4, to identify the common bit positions suitable for splitting (line 5). Once the secure summation step is finished the resulting globally summed ratio vector V_G is used to find the n_b best splitting bit positions by using the function *getCombinations()* (in line 6). In this function, first the bit positions with a globally summed ratio value less than the bit selection threshold bs_t are selected into the set I of splitting bit positions as shown in Equation 6.2. The n_b bit positions in I with the lowest ratio values (which we call *match-bits*) are selected as the best splitting bit positions. Fig. 6.3 illustrates an example of selecting the best splitting bits.

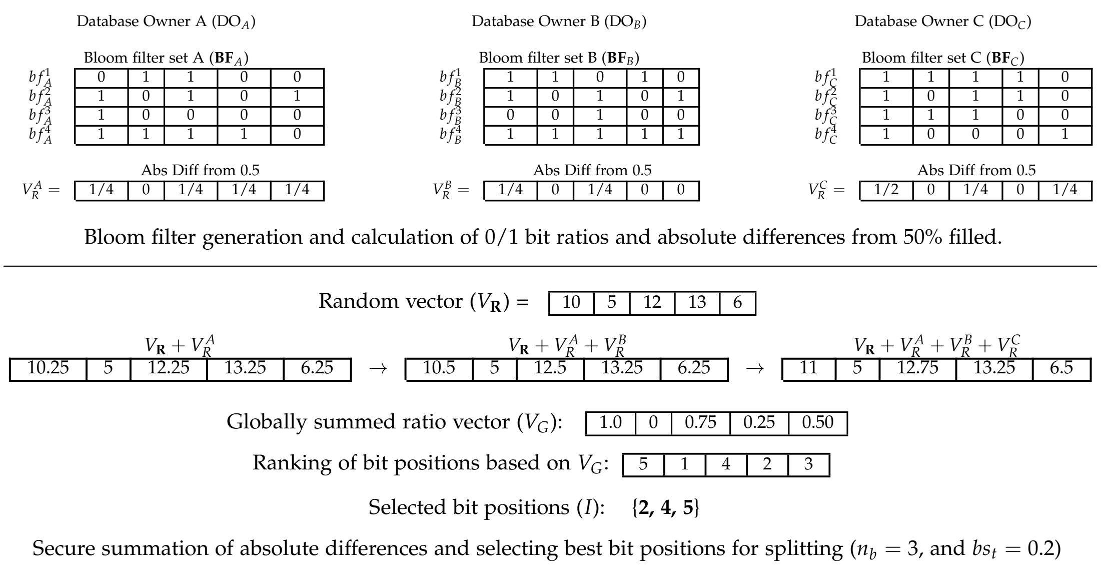
$$I = \{j \mid (\frac{d}{2} - \sum_{i=1}^{d} f_{ij}) < bs_t\} \tag{6.2}$$

Algorithm 6.1: Multi-bit Bloom filter splitting by DO_i, with $1 \leq i \leq d$

```
Input  : BF_i     - An inverted index of BFs belonging to DO_i
         mb_min   - Minimum size of a mini-block
         mb_max   - Maximum size of a mini-block
         n_b      - Maximum number of bit positions for splitting
         bs_t     - Bit selection threshold
Output: MB_i  - Set of mini-blocks
Q ←[BF_i]                                           // Initialisation of queue
while Q ≠ [] do
    B ← Q.pop()                                     // Get the current block
    V_R ← genRatios(B)                              // Generate local bit ratios
    V_G ← secureSummation(V_R)                      // Get ratios globally
    BC ← getCombinations(V_G, n_b, bs_t)            // Get bit combinations
    V_C ← getCombSuitability(B, BC, mb_min)         // Check bit combinations suitability
    V_G ← secureSummation(V_C)                      // Perform a secure summation
    bc_G ← getGlobalCommbination(V_G,BC)            // Get globally best bit combinarion
    if bc_G ≠ -1 then                               // Check if a globally best bit combination is available
        MB ← splitBlock(bc_G, B)                    // Compute mini-blocks
        foreach mb ∈ MB do
            if |mb| ≥ mb_max then                   // Check if the mini-block is too large
                Q.push(mb)                          // Add to queue for further splitting
            end
            else
                MB_i ∪ mb                           // Add the mini-block to the final set of mini-blocks
            end
        end
    end
    else
        MB_i ∪ B                                    // Add current block to the set of mini-blocks
    end
end
return MB_i
```

Based on the selected bit positions the function *getCombinations()* returns the set of all possible bit combinations BC. As shown in Figure 6.3, bit positions $\{2, 4, 5\}$ would generate the set of combinations of $\{\{2, 4, 5\}, \{2, 4\}, \{2, 5\}, \{4, 5\}, \{2\}, \{4\}, \{5\}\}$. It is important to consider every possible bit combination across these selected bit positions for splitting and select the most suitable combination, as we describe below. This is because some bit combinations could potentially split B into a set of mini-blocks that do not satisfy the required size bounds (from mb_{min} to mb_{max}).

For each combination in BC, the BFs in the current block B are counted to analyse the sizes of resulting mini-blocks under different bit patterns by the function *getCombSuitability()* in line 7. For example, for the bit combination $\{2, 4\}$ in BC, the sets of bit patterns considered are $\{00, 01, 10, 11\}$. Once the current block is processed for all possible bit combinations, the function *get-*



Database Owner A (DO_A)

Bloom filter set A ($\mathbf{BF}_A$)

bf_A^1	0	1	1	0	0
bf_A^2	1	0	1	0	1
bf_A^3	1	0	0	0	0
bf_A^4	1	1	1	1	0

Abs Diff from 0.5

$V_R^A =$	1/4	0	1/4	1/4	1/4

Database Owner B (DO_B)

Bloom filter set B ($\mathbf{BF}_B$)

bf_B^1	1	1	0	1	0
bf_B^2	1	0	1	0	1
bf_B^3	0	0	1	0	0
bf_B^4	1	1	1	1	1

Abs Diff from 0.5

$V_R^B =$	1/4	0	1/4	0	0

Database Owner C (DO_C)

Bloom filter set C ($\mathbf{BF}_C$)

bf_C^1	1	1	1	1	0
bf_C^2	1	0	1	1	0
bf_C^3	1	1	1	0	0
bf_C^4	1	0	0	0	1

Abs Diff from 0.5

$V_R^C =$	1/2	0	1/4	0	1/4

Bloom filter generation and calculation of 0/1 bit ratios and absolute differences from 50% filled.

Random vector ($V_\mathbf{R}$) = | 10 | 5 | 12 | 13 | 6 |

$V_\mathbf{R} + V_R^A$	10.25	5	12.25	13.25	6.25
$\rightarrow$ $V_\mathbf{R} + V_R^A + V_R^B$	10.5	5	12.5	13.25	6.25
$\rightarrow$ $V_\mathbf{R} + V_R^A + V_R^B + V_R^C$	11	5	12.75	13.25	6.5

Globally summed ratio vector (V_G):	1.0	0	0.75	0.25	0.50
Ranking of bit positions based on V_G:	5	1	4	2	3

Selected bit positions (I): **{2, 4, 5}**

Secure summation of absolute differences and selecting best bit positions for splitting ($n_b = 3$, and $bs_t = 0.2$)

Figure 6.3: Selecting best bit positions for multi-bit Bloom filter splitting

CombSuitability() outputs a binary vector V_C of length $|BC|$. Each bit value in V_C corresponds to a combination in BC. For a given combination in BC, if all the resulting mini-blocks have at least mb_{min} BFs then the corresponding bit in V_C is set to 1, otherwise it is set to 0. For bit positions $\{2, 4, 5\}$ in the example above, if only bit combinations $\{2, 4\}$ and $\{2\}$ generate the mini-blocks with mb_{min} BFs then V_C = [0, 1, 0, 0, 1, 0, 0], where the 1-bit at 2 and 5 correspond to $\{2, 4\}$ and $\{2\}$, respectively.

Once all the DOs generate their V_Cs, a secure summation is performed again to find the common best bit combination in line 8. Based on the globally summed vector V_G, the function *getGlobalCommbination* returns a bit combination as the globally accepted bit combination bc_G (line 9). In V_G the bit combination that has its corresponding bit in V_C equals to d (number of DOs) is selected as best combination bc_G. This ensures the selected bit combination could generate mini-blocks with at least mb_{min} BFs across all the DOs. However, if more than one combination could be selected as a possible bc_G then the combination with most number of bits is selected for the splitting. Such a selection approach ensures in each splitting iteration B is split into as many mini-blocks as possible. Following the same example above, $\{2, 4\}$ is selected as bc_G from the list of suitable combinations [$\{2, 4\}$, $\{2\}$]. B is split into a set of mini-blocks (MB) according to the selected bc_G by using function *splitBlock()* (line 11).

In line 10, if neither of the combinations in BC can be used to split B (bc_G = -1), then the current block B is added to a set **MB** (line 22). All the mini-blocks in MB that contain more than mb_{max} BFs are added back into the queue Q for further splitting (lines 13 and 14). The parameters mb_{min} and mb_{max} allow the DOs to control the sizes of the mini-blocks generated in Algorithm 6.1.

4. **Step 4 - Mini-block Merging**

   Once the set of mini-blocks **MB** is generated, each DO merges similar mini-blocks into larger blocks based on their similarities to facilitate computationally efficient blocking. We use a canopy clustering technique [48, 144] to merge these mini-blocks together. In our merging step we consider each mini-block generated by the multi-bit splitting algorithm as a separate cluster.

   To merge mini-blocks we first select a BF bf_k as a medoid of each cluster. This BF bf_k is selected based on the Hamming distances between the BFs in a mini-block mb that has the minimum distance to all the other BFs in mb. For this selection process we suggest a maximum average Hamming distance based similarity calculation which is shown in Equation 6.3.

$$bf_k = \underset{k}{\text{argmax}} \left\{ \frac{\sum_{k=1}^{|mb|} sim_H(bf_k, bf_l)}{|mb|} \mid bf_k, bf_l \in mb,\ 1 \leq k, l \leq |mb|,\ k \neq l \right\} \tag{6.3}$$

   The function $sim_H()$ computes the similarity of two BFs bf_i and bf_j by computing the normalised Hamming distance between them as shown in Equation 6.4:

$$sim_H(bf_k, bf_l) = 1 - \frac{x}{l_{bf}}, \tag{6.4}$$

   where x represents the number of bit positions that differ in bf_k and bf_l. For the merging of mini-blocks we suggest to use the following two canopy based clustering algorithms:

   4.a *Standard canopy clustering*: Mini-blocks are merged until the resulting block size increases to b_{min}. This algorithm allows to merge a set of similar mini-blocks greedily in a given iteration.

   4.b *Hierarchical canopy clustering*: The merging of mini-blocks is based on an agglomerative clustering approach. In a given iteration, the two most similar mini-blocks are merged, thereby guaranteeing that the sizes of all the resulting blocks have a lower bound of b_{min}.

   The blocks generated by these clustering approaches can be compared using an appropriate private comparison and classification technique later in the PPRL pipeline [212, 213]. We next describe both these clustering algorithms in more detail.

   4.a **Merge mini-blocks with standard canopy clustering**

   The suggested standard canopy clustering algorithm for the merging of mini-blocks is shown in Algorithm 6.2. In line 2, each DO$_i$ iterates over its set of mini-blocks **MB**$_i$. At each iteration the mini-block mb_f at the beginning of **MB** is processed as the initial cluster (line 3). The function *getMedoid()* computes the medoid c_f of mini-block mb_f (line 4). This function uses Equation 6.3 to compute medoids.

Algorithm 6.2: Merge mini-blocks using standard canopy clustering

```
Input  : $\mathbf{MB}_i$ - Set of mini-blocks of $DO_i$
         $ts_t$ - Tight similarity threshold
         $ls_t$ - Loose similarity threshold
         $b_{min}$ - Minimum size of merged block
Output: $\mathbf{B}_i$ - Set of merged blocks
1  $\mathbf{B}_i \leftarrow \emptyset$                          // Initialisation of the set of merged blocks
2  while $\mathbf{MB}_i \neq \emptyset$ do
3      $mb_f \leftarrow \mathbf{MB}_i.pop()$                    // Get the frist mini-block
4      $c_f \leftarrow getMedoid(mb_f)$                         // Compute the centroide of the mini-block
5      $B \leftarrow mb_f$                                      // Initialise the merging block
6      while $|B| < b_{min}$ do                                 // Merge blocks until the minimum block size is met
7          $mb_k \leftarrow \mathbf{MB}_i.next()$               // Get the next mini-block
8          $c_k \leftarrow getMedoid(mb_k)$                     // Compute the medoids of the mini-block
9          $s \leftarrow sim_H(c_f, c_k)$                       // Compute the Hamming distance between the medoids
10         if $s \geq ts_t$ then                                // Check if tight similarity threshold is met
11             $B \leftarrow B \cup mb_k$                       // Merge the two mini-blocks
12             $\mathbf{MB}_i.delete(mb_k)$                     // Remove the merged mini-block
13         end
14         else if $s \geq ls_t$ then                           // Check if loose similarity threshold is met
15             $B \leftarrow B \cup mb_k$                       // Merge the mini-blocks
16         end
17     end
18     $\mathbf{B}_i.add(B)$                                    // Add the merged block to the set of merged blocks
19 end
20 return $\mathbf{B}_i$
```

The initial cluster mb_f is compared and merged with all other mini-blocks in **MB** until the size of the merged block B reaches b_{min} as in lines 6 to 18. For each mini-block mb_k in $\mathbf{MB}_i$ a medoid c_k is computed in line 8. Next, a Hamming similarity value s is computed using Equation 6.4 between medoids c_f and c_k (line 9).

At the merging step, the computed similarity value s between the medoids is checked against the tight and loose thresholds ts_t and ls_t, with $ts_t \geq ls_t$. As per lines 10 to 13, if s is equal to or greater than the tight threshold ts_t then the mini-block mb_k is merged with mb. Next, the merged mini-blocks are removed from the set $\mathbf{MB}_i$ to avoid them being considered again as clusters in a following iteration (line 12). If s is equal to or greater than the loose threshold ls_t only the initial cluster mb is removed from the set **MB** (lines 14 to 16) and the other mini-blocks (that were merged with mb) are left in **MB** to be used again in the following iterations.

Once the size of the resulting merged block B reaches b_{min}, B is added to the set of merged blocks $\mathbf{B}_i$ in line 18. Therefore at each iteration a set of mini-blocks which are similar to the initial cluster mb are merged together until the resulting block size b_{min} is reached.

Algorithm 6.3: Merge mini-blocks using hierarchical canopy clustering

```
Input  : $\mathbf{MB}_i$ - Set of mini-blocks of $DO_i$
         $ts_t$ - Tight similarity threshold
         $b_{min}$ - Minimum size of merged block
Output: $\mathbf{B}_i$ - Set of merged blocks
$\mathbf{B}_i \leftarrow \emptyset$                    // Initialisation of the set of merged blocks
while $\mathbf{MB}_i \neq \emptyset$ do
    $mb_f \leftarrow \mathbf{MB}_i.pop()$             // Get the frist mini-block
    $c_f \leftarrow getMedoid(mb_f)$                  // Compute the medoid of the mini-block
    $mb_k, c_k \leftarrow getMiniBlock(c_f)$          // Get the mini-block for merging
    $B \leftarrow mb_f \cup mb_k$                     // Merge the two mini-blocks
    if $sim_H(c_f, c_k) \geq ts_t$ then               // Check if tight similarity threshold is met
        $\mathbf{MB}_i.delete(mb_k)$                  // Remove the merged mini-block
    end
    if $|B| < b_{min}$ then                           // Check the merged block size
        $\mathbf{MB}_i.add(B)$                        // Add the merged block back to $\mathbf{MB}_i$
    end
    else
        $\mathbf{B}_i.add(B)$                         // Add the merged block to the set of merged blocks
    end
end
return $\mathbf{B}_i$
```

4.b **Merge mini-blocks with hierarchical canopy clustering**

The standard canopy clustering approach described above merges a set of similar mini-blocks in a given iteration until the resulting merged block reaches the minimum block size b_{min}. However, due to this greedy merging of mini-blocks the size of a merged block can go beyond the b_{min} size limit which results in a large number of candidate record tuples to be generated. Also, in the merging process mini-blocks are merged iteratively if they satisfy the tight and loose similarity thresholds ts_t and ls_t. This could result in a mini-block that is highly similar to the initial mini-block selected not to be merged if the merged block already has reached b_{min}.

To address this issue we propose a novel threshold based hierarchical agglomerative canopy clustering approach which merges the two most similar mini-blocks in a given iteration until the acceptable block size is reached. Our mini-block merging algorithm based on hierarchical canopy clustering is shown in Algorithm 6.3.

To merge mini-blocks each DO iterates over the set of mini-blocks **MB** (line 2). At each iteration, a mini-block mb_f (initial cluster) is selected and the medoid c_f is computed using Equation 6.3 (lines 3 and 4). The function *getMiniBlock()* computes the mini-block mb_k that is most similar to mb_f using Equation 6.5, and returns with its medoid c_k (line 5).

$$mb_k = \underset{j}{\operatorname{argmax}} \{sim_H(c, c_k) \mid \forall (mb, mb_k) \in \mathbf{MB},\ 1 \leq k \leq |\mathbf{MB}|\} \tag{6.5}$$

The selected mini-block mb_k is merged with mb_f into a new block B (line 6). If the similarity between the medoids c_f and c_k is greater than or equal to the tight similarity threshold ts_t then the mini-block mb_k is removed from the set of mini-blocks **MB** (lines 7 to 9).

Finally, the size of the resulting merged block B is checked against the size limit b_{min} (lines 10 to 12). If the size of B is less than b_{min}, then B is added into **MB** as a new mini-block for further merging (line 11). This enables B to be merged further with other similar mini-blocks.

In the next iteration, a new medoid is computed for B based on all the BFs grouped in the previous merging of mini-blocks. This means only the two most similar mini-blocks are merged in each iteration. B is added into the final set of merged blocks **B** once it reaches b_{min} (line 14).

6.4 Conceptual Analysis of Clustering based Blocking

In this section we analyse our private blocking approach in terms of complexity, privacy, and quality of blocking.

6.4.1 Complexity

We analyse the computational and communication complexities of our blocking approach in terms of a single DO. Let us assume there are $|\mathbf{D}| = n_r$ records in a database with each containing an average of n_q q-grams. In the second step of our approach, as shown in Figure 6.2, all records are encoded using n_h hash functions. The Bloom filter generation for a single DO therefore has a complexity of $O(n_h \cdot n_q \cdot n_r)$.

In the multi-bit splitting step (step 3 in Figure 6.2), the parameters mb_{min} and mb_{max} are used to control the size of mini-blocks generated. The sizes of mini-blocks decide the number of iterations that are required in the splitting phase. At each iteration a given block B is split into $|MB| = 2^{n_b}$ mini-blocks if n_b bit positions are selected for splitting. The number of iterations in the splitting phase can be calculated as $n_i = log(n_r/mb_{max})/log(|MB|)$. Therefore the splitting of n_r Bloom filters into a set of mini-blocks is of $O(n_r \cdot n_i)$.

In the forth step of our approach, merging mini-blocks requires the processing of $|\mathbf{MB}|$ mini-blocks. The computation of the medoids for all mini-blocks is of $O(mb_{max}^2 \cdot |\mathbf{MB}|)$ complexity because for each mini-block in **MB** a medoid needs to be computed. Merging mini-blocks using the standard canopy clustering technique requires a total computation of $O(\frac{b_{min}}{mb_{max}} \cdot |\mathbf{B}|)$, where at each iteration $\frac{b_{min}}{mb_{max}}$ clusters are merged. At each iteration of the hierarchical canopy clustering approach the two most similar mini-blocks are merged which requires a total of $O(|\mathbf{B}|^2)$ computations.

The DOs only need to communicate with each other to perform the secure summation protocol. By assuming each of the d DOs is directly connected to all other DOs, d messages each of size l_{bf} need to be exchanged in each iteration. Therefore the entire clustering based blocking approach has a communication complexity of $O(d \cdot n_i)$ for d database owners.

6.4.2 Blocking Quality

We analyse the quality of the multi-bit splitting (step 3 in Figure 6.2) and merging (step 4 in Figure 6.2) processes of our blocking approach in terms of effectiveness and efficiency [215]. In multi-bit splitting the selection of bit positions is computed globally based on the BFs across all the DOs. However, due to the splitting process similar BFs in a given database could be grouped into different mini-blocks, since each DO has different $1/0$ ratios in their current processed blocks in a given splitting iteration. This could potentially decrease the quality of blocking if no merging is used later in our approach. However, using multiple bits rather than a single bit in each splitting iteration increases the efficiency of the overall blocking process because in a given iteration a block is split into multiple smaller blocks instead of only two blocks.

In step 4 of our approach, the standard canopy clustering algorithm merges mini-blocks greedily, making the block generation process more efficient compared to the hierarchical canopy clustering algorithm. Hierarchical canopy clustering merges mini-blocks in an agglomerative manner which potentially requires more merging iterations. However, in each merging iteration the hierarchical canopy clustering algorithm merges the two most similar mini-blocks while in standard canopy clustering mini-blocks are merged if they satisfy the similarity thresholds. Furthermore, once two mini-blocks are merged a new mediod is computed to calculate the similarity between other mini-blocks in the hierarchical canopy clustering technique. This ensures BFs that have the lowest Hamming distance (highest similarity) are grouped together in each iteration (as detailed in Algorithm 6.3). Therefore, hierarchical canopy clustering can group highly similar BFs together, while in standard canopy clustering it is more likely that two highly similar mini-blocks would not be merged due to the sequential merging process of Algorithm 6.2.

However, both clustering approaches could potentially generate blocks with a number of records larger than b_{min}. For $|\mathbf{B}|$ merged blocks, with each containing b_{min} records, the number of candidate record tuples (CRTs) generated for d DOs is $(b_{min})^d \cdot |\mathbf{B}|$. In the merging process the parameter b_{min} limits the sizes of the blocks generated by the clustering algorithms which indirectly determines the number of merged blocks generated by the blocking approach. However, in the worst case scenario a merged block could contain $2(b_{min} - 1)$ records if the two mini-blocks that are to be merged are of size $b_{min} - 1$. Therefore, an appropriate value for b_{min} needs to be set by the DOs considering factors such as their database size and the number of DOs that are participating in blocking because more messages need to be communicated in the merging process if b_{min} is set to a larger value (more merging iterations). We leave the work on how to calculate optimal values for parameters mb_{min}, mb_{max}, and b_{min} as a future research.

6.4.3 Privacy

Privacy needs to be considered thoroughly to assess the amount of information a DO can learn from the data it receives from the other DOs when they communicate among each other. We assume each DO follows the honest-but-curious (HBC) adversary model [212], as described in Section 2.4 on page 27.

In our blocking approach each DO participates in a secure summation protocol for exchanging of ratio values of BFs with other DOs as described in Section 6.3. During these summations, each DO computes their ratio values but neither of the DOs is capable of deducing anything about any other DOs' private inputs since the random vector $V_{\mathbf{R}}$ is only known to the DO that initiated the protocol. However, as we described in Section 5.4 on page 101, DOs are susceptible to collude with each other to learn about data of a non-colluding DO. To address this a collusion resistant secure summation protocol (as described in Section 4.3 on page 70) can be used.

Our approach performs a generalisation strategy on the blocks by merging mini-blocks together to generate larger blocks. The parameter b_{min} is used to guarantee that every resulting merged block contains at least b_{min} records. This ensures all the blocks generated have the same minimum number of records, which guarantees k-anonymous ($k = b_{min}$) privacy [110, 215]. The merging of mini-blocks to create larger blocks makes this blocking approach less vulnerable to a dictionary or frequency attack [212] because all the blocks contain the same minimum number of records which gives less information about the frequency distributions of attribute values to an attacker. A higher value for b_{min} provides stronger privacy guarantees but would require more computations as more candidate record tuples (CRTs) will be generated. Therefore the parameter b_{min} needs to be chosen carefully.

One main aspect of canopy clustering is that it can generate overlapping clusters. In the PPRL context overlapping clusters can disclose information about the records that appear in multiple blocks. The parameters ts_t and ls_t need to be adjusted appropriately in order to keep the overlap between the generated blocks at a minimum. In both clustering techniques if $ts_t = ls_t$ then there would not be any overlap present among the generated blocks because in each iteration all merged mini-blocks are removed to avoid them being considered again as clusters [35]. This ensures each BF will only be inserted into one block. However, such a setting might reduce the overall linkage quality because each block is compared only with its corresponding blocks from other DOs and similar BFs that are assigned in different blocks will not be compared. Therefore, ls_t needs to be set smaller and closer to ts_t to increase the linkage quality, but also to keep overlapping blocks at a minimum. We leave the development of a method to calculate the optimal values for ts_t and ls_t in terms of overlapping and linkage quality as a future work.

Table 6.2: Parameter settings used in the experimental evaluation

Parameter	Value range
Dataset name	NC-SYN
Number of databases (d)	3, 5, 7, 10
Number of records in each database (n_R)	5,000 to 1,000,000
Blocking key attributes (A)	Given name, Surname, City (suburb), and Postcode
Corruption levels	0%, 20%, and 40%
Bloom filter length (l_{bf})	1000
Number of hash functions (n_h)	30
Character length of q-gram (n_q)	2
Minimum mini-block size (mb_{min})	10
Maximum mini-block size (mb_{max})	20
Number of splitting bit positions (n_b)	1, 2, 3
Bit selection threshold (bs_t)	0.1
Tight similarity threshold (ts_t)	0.9
Loose similarity threshold (ls_t)	0.8
Minimum merged block size (b_{min})	50, 100, 500, 1000

Once the blocks are generated, we assume the DOs use an appropriate private comparison and classification technique [210] to match *CRT*s, which must not reveal any information regarding the sensitive attributes and non-matches. However, the comparison and classification of *CRT*s is not a step in our blocking approach.

6.5 Experimental Evaluation

In this section, we present and discuss the results of the experimental evaluation study of our clustering based blocking approach conducted on the datasets described in Section 4.4 (on page 78). Table 6.2 summarises the parameter values used in our approach. In this section we use the terms MBS, SCC, and HCC to indicate multi-bit splitting, standard canopy clustering, and hierarchical canopy clustering, respectively.

6.5.1 Scalability

We measured the average and total runtime of our clustering based blocking approaches to evaluate the complexity. Figure 6.4 (a) shows the scalability of steps 1 and 2 of our approaches in terms of database sizes. As expected the runtime required for Bloom filter (BF) generation increases linearly with database size as more time is required to encode records into BFs. As can be seen in this figure, the efficiency of the multi-bit splitting step depends on the number of splitting bit (n_b) selection. The splitting step requires less runtime when n_b increases because more mini-blocks are generated in each iteration which reduces the overall number of splitting iterations required by Algorithm 6.1.

As shown in Figure 6.4 (b), the total runtime required for blocking increases with the number of databases because more messages are communicated between the participating DOs during the multi-bit splitting and clustering steps. However,

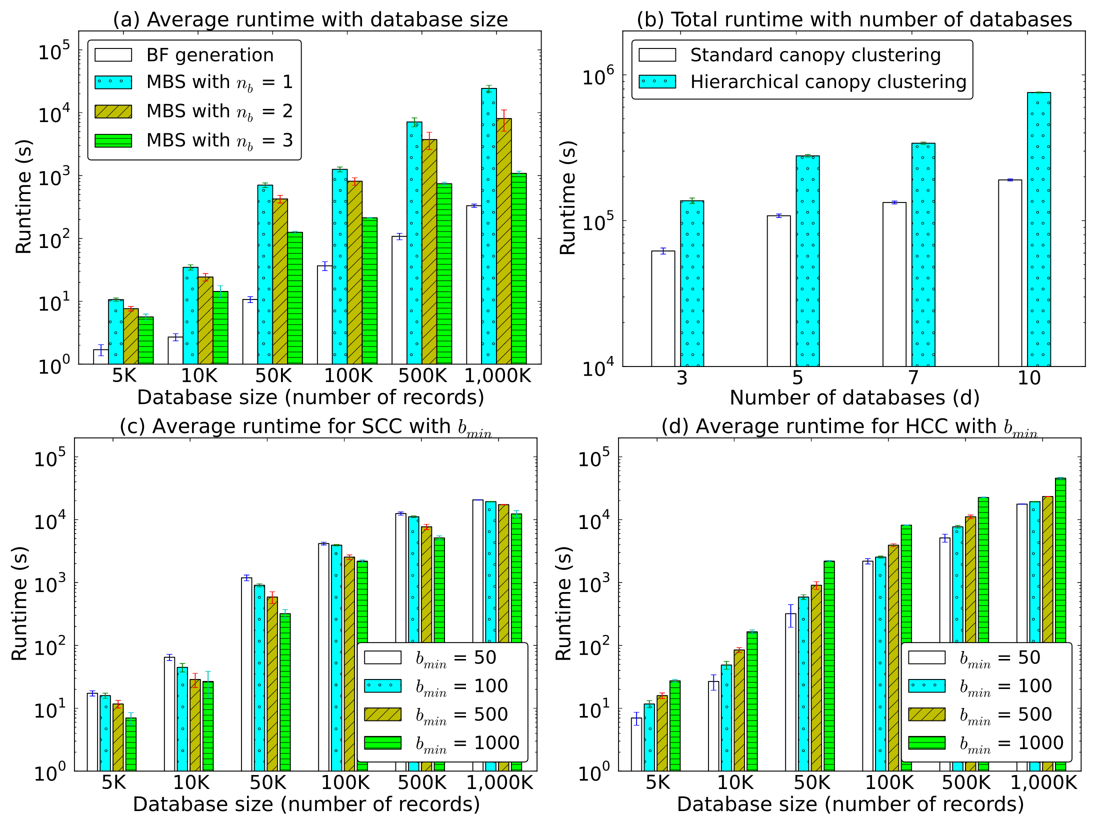


Figure 6.4: (a) Average runtime for BF generation and multi-bit splitting with different minimum number of splitting bits, (b) total runtime with different number of databases, and average runtime for (c) standard canopy clustering and (d) hierarchical canopy clustering. Note that plots have different y-axis scales.

blocking with standard canopy clustering (SCC) consumes less runtime compared to hierarchical canopy clustering (HCC) since more merging iterations are required in HCC than SCC.

Figures 6.4 (c) and (d) show the average runtime required for our blocking approach with SCC and HCC for different minimum block sizes (b_{min}), respectively. As shown in Figure 6.4 (c), for a larger b_{min} the runtime required by SCC decreases because more mini-blocks are merged in each iteration which reduces the overall merging iterations required by the SCC technique. As shown in Figure 6.4 (d), the average runtime required by HCC increases with b_{min} because more iterations are required due to its hierarchical merging approach. However, both SCC and HCC show a linear scalability with the size and number of the databases that are blocked. We also measured the average memory required by our blocking approach, which increases with the database size. However, it consumed just over 1 GByte of memory for blocking a database with 1 million of records.

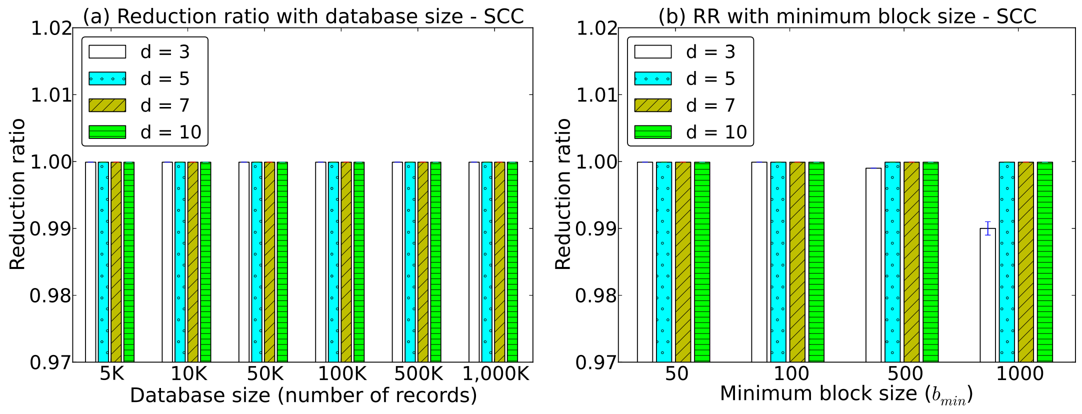


Figure 6.5: Reduction ratio (RR) of our blocking approach with standard canopy clustering (SCC) technique for different (a) database sizes and (b) minimum block sizes (b_{min}).

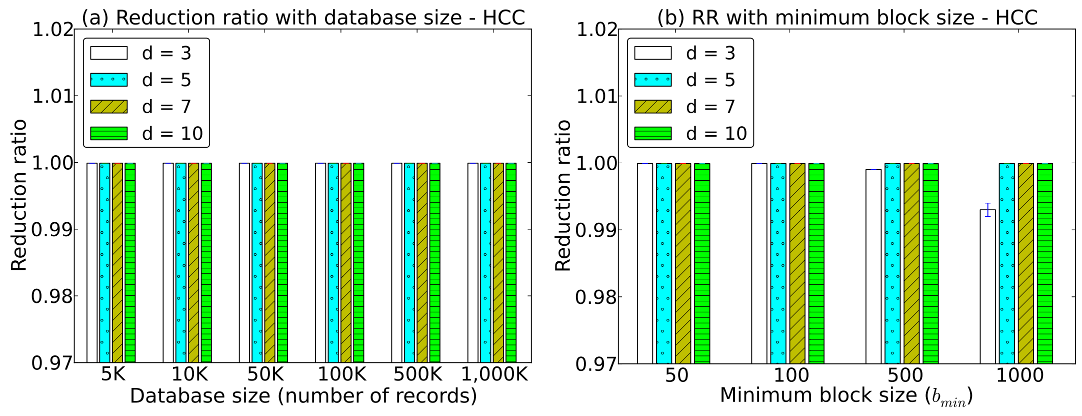


Figure 6.6: Reduction ratio (RR) of our blocking approach with hierarchical canopy clustering (HCC) for different (a) database sizes and (b) minimum block sizes (b_{min}).

We also measured the scalability of our blocking approach in terms of the adapted reduction ratio (RR) measure described in Section 2.5 on page 35. As can be seen in Figures 6.5 (a) and 6.6 (a), RR remains close to 1 for different number of databases for both of the clustering techniques. This indicates that our clustering based blocking approach reduces the number of total candidate record tuples (*CRT*s) that need to be compared dramatically. The number of *CRT*s that need to be compared is only around 0.0001% of the total number of possible record tuple comparisons across all databases to be linked. Similar to our tree based blocking approach described in Chapter 5, such a reduction is achieved because only the corresponding blocks across all the blocked databases are compared which limits the number of candidate block tuples to be compared (see also Table 5.4 on page 105). A similar behaviour can also be seen with different values for b_{min} where RR remains closer to 1 even when b_{min} is increased, as shown in Figures 6.5 (b) and 6.6 (b).

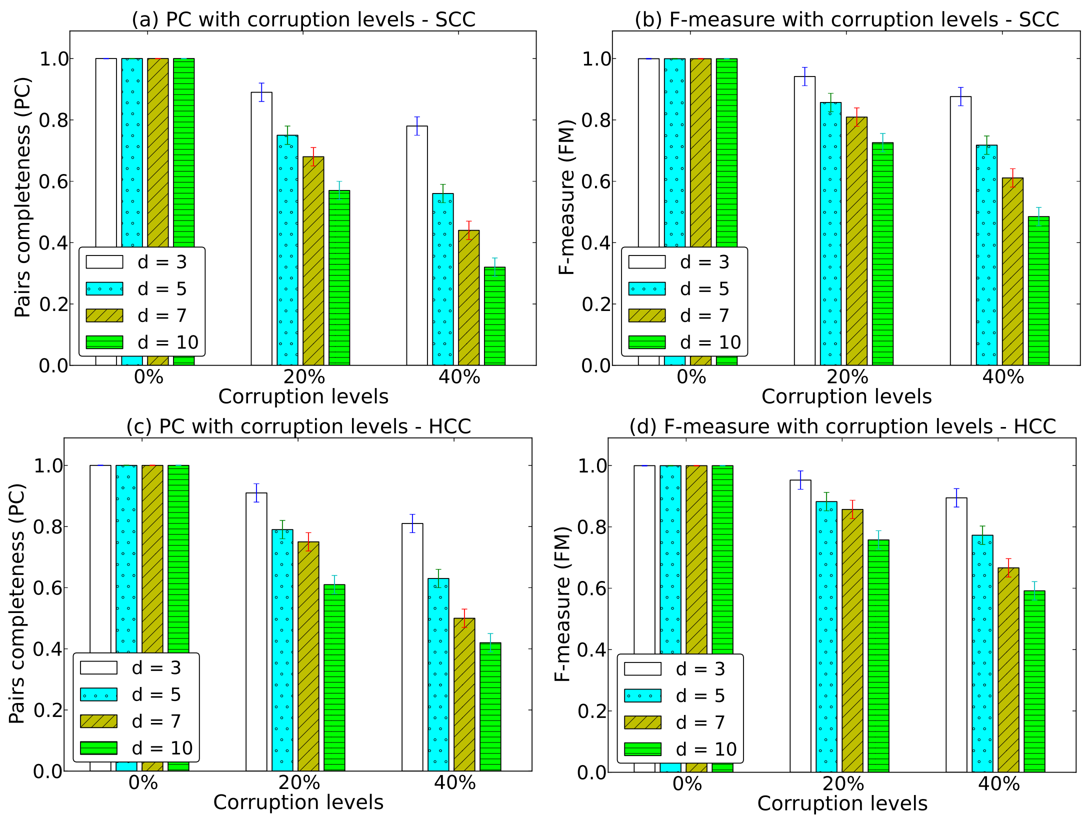


Figure 6.7: (a) and (c) Pairs completeness (PC), and (b) and (d) F-measure (FM) with different number of databases for different corruption levels for standard canopy clustering (SCC) and hierarchical canopy clustering (HCC), respectively.

6.5.2 Blocking Quality

We measure the blocking quality of our blocking approach in terms of pairs completeness (PC) and F-measure (FM), as described in Section 4.4 on page 83. We used the corruption levels 0%, 20%, and 40% of the NC-SYN dataset (as detailed in Section 4.4 on page 78) to evaluate the blocking quality of our approach.

As illustrated in Figures 6.7 (a) and (c), our approach achieves a PC of 1 for 0% corruption with different number of databases for both SCC and HCC. As can be seen in these figures, both clustering approaches achieve acceptable level of PC even in the presence of dirty data. As we expected our blocking approach achieves higher PC rates with HCC compared to SCC because in each iteration the two most similar mini-blocks are merged. This improves the overall blocking quality as more similar BFs are grouped together. Figures 6.7 (b) and (d) show that the F-measure (FM) also decreases with the corruption level as the number of databases to be blocked increases.

However, we noted that PC decreases with the number of databases which suggests that similar BFs are grouped into different blocks due to the collaboration of DOs in the multi-bit splitting and merging processes. The presence of dirty data

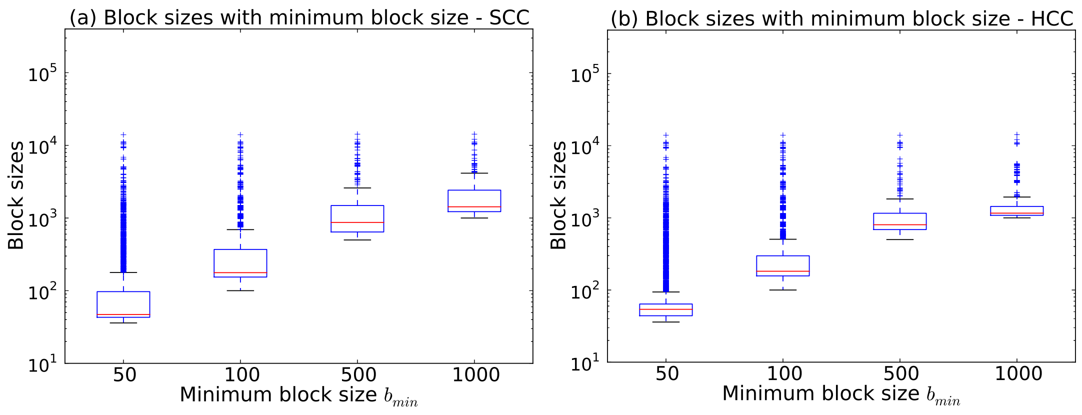


Figure 6.8: Average block sizes with minimum block size (b_{min}) for (a) standard canopy clustering (SCC) and (b) hierarchical canopy clustering (HCC) for a 1,000K database. K represents 1,000 records.

in these databases effects the bit selection process in multi-bit splitting as well as the selection of the mini-blocks in the merging process. However, as can be seen in Figure 6.7, the merging of mini-blocks using HCC improves the blocking quality of our approach which suggests HCC is more suitable compared to SCC for blocking databases with dirty data.

6.5.3 Privacy

To evaluate the privacy of our approach we studied the block sizes generated for different minimum block size (b_{min}) values, as shown in Figures 6.8. These figures illustrate that both clustering approaches generate blocks that contain at least b_{min} records ensuring k-anonymous privacy where k =b_{min}. However, as can be seen in Figure 6.8 (b), we noted that the variance between block sizes generated by HCC is lower compared to SCC (see Figure 6.8 (a)). This suggests that HCC can be used for block generation in scenarios where more control is needed in their sizes. However, we also noted that for smaller b_{min} (when b_{min} = 50) both SCC and HCC could result many blocks with large number of records ($> b_{min}$). We noted that many of these large blocks occur in the multi-bit splitting step where they are not processed in the merging step since these blocks already satisfy the minimum block size. This could potentially increase the overall number of CRTs to be compared in the comparison step of the PPRL process. Therefore, based on the parameters mb_{min} and mb_{max} a DO might need to set a suitable value for b_{min}.

We computed average probability of suspicion (PS) values (for the frequency linkage attack described in Section 2.5 on page 38) for a database with 1 million records for different b_{min}. As shown in Figure 6.9, our blocking approach results in a maximum PS value less than $(1/b_{min})$ for each b_{min} value, which indicates a record in a block can be matched to more than b_{min} values in a global database $\mathbb{G}$ (under the worst case assumption of $\mathbb{G}$ is being equal to the blocked database). As shown

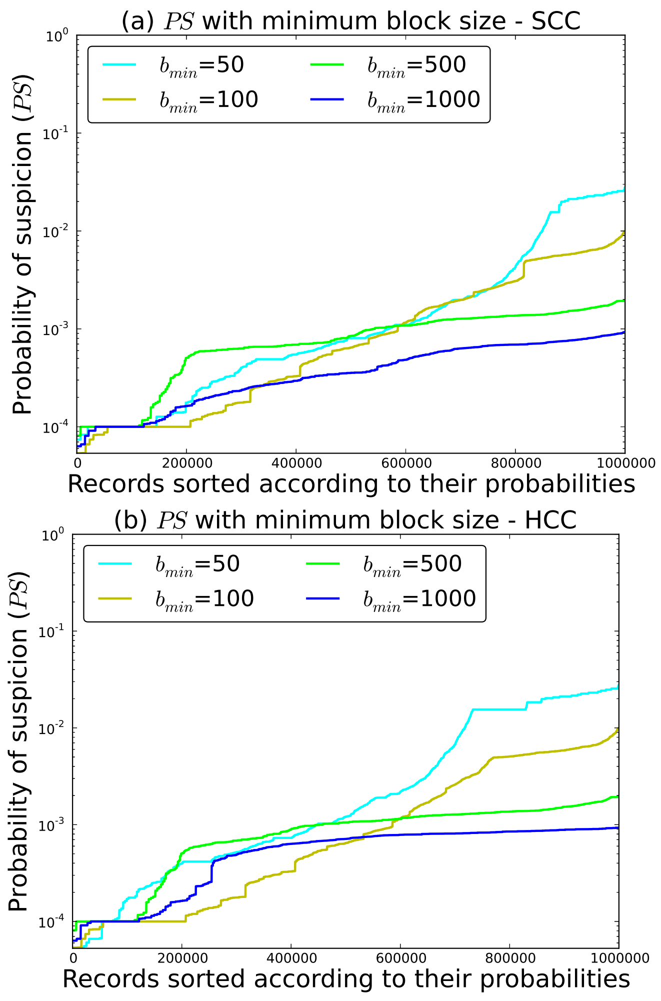


Figure 6.9: Probability of suspicion (PS) with minimum block size (b_{min}) for (a) standard canopy clustering (SCC) and (b) hierarchical canopy clustering (HCC) for a 1,000K database. In here K represents 1,000 records.

in Figure 6.9, HCC provides better privacy compared to SCC because most of the generated blocks are within the acceptable size limit.

We also measured the average (DS_{Mean}), median (DS_{Med}), and maximum (DS_{Max}) disclosure risk values with different b_{min} values (as we described in Section 4.4). As shown in Figures 6.10 (a) and (b), the DR values decrease for larger block sizes as more records are included in a given block making our blocking approach more secure against frequency attacks. As can be seen, SCC has lower DS_{Med} values for every b_{min} compared to HCC because more larger blocks are generated by SCC.

Finally, we conducted the cryptanalysis attack we described in Section 4.5 on page 85. We evaluate the accuracy of our attack by calculating (1) the percentage of correct guesses with 1-to-1 matching, (2) the percentage of correct guesses with 1-to-m (many) matching, (3) the percentage of wrong guesses, and (4) the percentage of no guesses, where these four percentages sum to 100. These four categories are

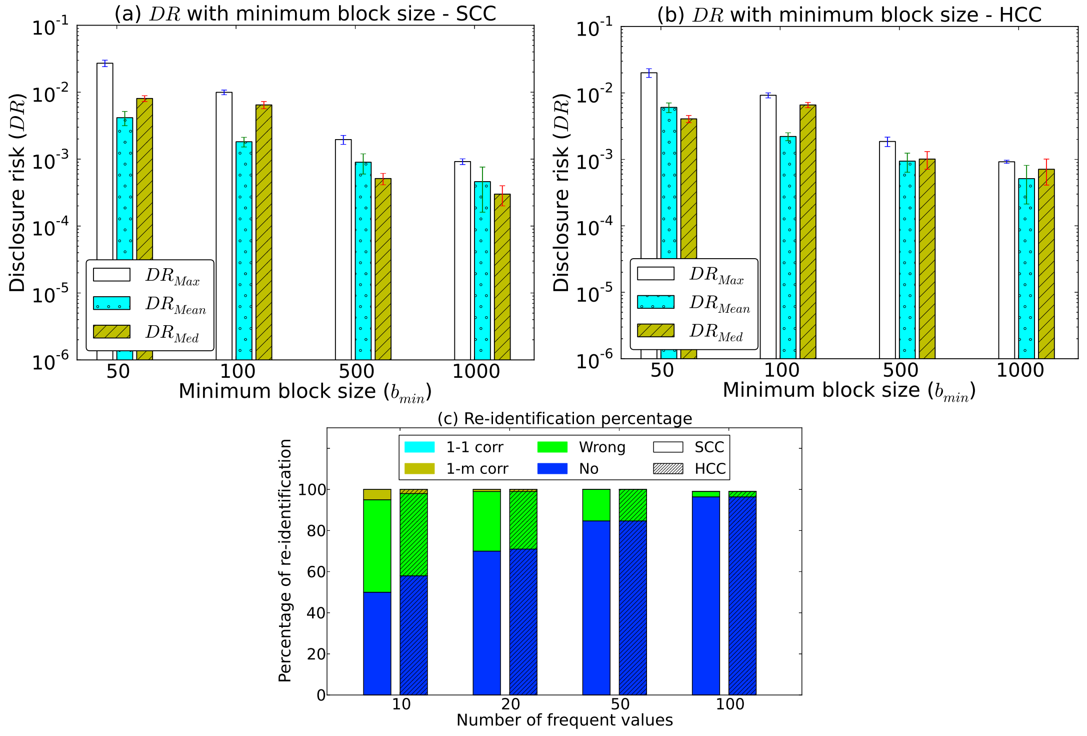


Figure 6.10: Disclosure risk values with minimum block size (b_{min}) for (a) standard canopy clustering (SCC) and (b) hierarchical canopy clustering (HCC), and (c) the cryptanalysis attack results performed upon the generated blocks for a 1,000K database. K represents 1,000 records.

labelled as *1-1 corr*, *1-m corr*, *Wrong*, and *No* in Figure 6.10 (c), respectively. We also set the parameters b_{min} and blocking key attribute (A) to $1,000$ and *Given name*, respectively.

Similar to Section 5.5 on page 107, we conducted the attack for each generated block of a database with 1 million records. As illustrated in Figure 6.10 (c), for both SCC and HCC most of the guesses result in no guesses. This indicates that an attacker could not re-identify an attribute value of *Given name* as each block does not contain enough frequency information to identify q-grams that are encoded in BFs. Therefore, conducting such cryptanalysis to correctly re-identify attribute values at the blocking level is impossible.

6.6 Summary

In this chapter we proposed a blocking approach for MD-PPRL based on multi-bit Bloom filter splitting and canopy clustering. We also suggested an agglomerative hierarchical canopy clustering algorithm which generates blocks (clusters) within a specific size range. We demonstrated the efficiency and effectiveness of our approach on large datasets containing up-to one million records. The evaluation results indicated that our approach is scalable with both the size and the number of databases.

As mentioned earlier, the number of splitting bit positions (n_b), the minimum (mb_{min}) and maximum (mb_{max}) mini-block size used in the multi-bit splitting step, and b_{min} used during clustering generate a trade-off between efficiency, effectiveness, and privacy of our clustering based blocking approach. As future work, we aim to investigate how to calculate optimal values for these parameters to improve the quality of blocking without compromising privacy. Parallelisation of Algorithms 6.1, 6.2, and 6.3 is another possible future research avenue of this approach.

Similar to our tree based blocking approach, one limitation in this approach is that the generation of blocks depends upon the collaborative participation of the DOs which requires frequent communication among them. To overcome this issue, in the next chapter we aim to investigate how to allow DOs to block their databases independently without frequent communication, and once blocks are generated locally by the DOs how to identify similar blocks across DOs that need to be compared.

Chapter 7

A Distributed Blocking Scheme for Multidatabase Privacy-preserving Record Linkage

The blocking techniques described in Chapters 5 and 6 require database owners to participate collaboratively to generate the blocks for their databases. In this chapter we propose a scalable distributed blocking scheme for multidatabase privacy-preserving record linkage (MD-PPRL). Our approach allows each database owner to perform blocking independently except for the initial agreement of parameter settings and a final central hashing-based clustering that efficiently and effectively prunes the record sets that are unlikely to match. In Section 7.1 we highlight the importance of distributed blocking in a multidatabase linkage context. In Section 7.2 we provide an overview of the proposed approach, and in Section 7.3 we describe the steps of our proposed approach in more detail. In Section 7.4, we provide a detailed analysis of our approach with regards to complexity, blocking quality, and privacy. In Section 7.5, we validate this analysis through an empirical experimental study. Finally, we summarise our findings in Section 7.6.

7.1 Introduction

As detailed in Chapter 1, the linkage of multiple databases across several database owners (DOs) can be performed by using two models. As illustrated in Figure 1.3 (on page 12) the linkage of different databases can be performed (1) only by the participation of DOs, or (2) with the participation of a linkage unit (LU) to facilitate the linkage.

Our blocking techniques described in Chapters 5 and 6 follow the first linkage model where only the DOs are participating in the blocking step to collaboratively block their own databases. However, such collaboration requires DOs to agree on the same parameter settings in the block generation process. Furthermore, our tree and clustering based blocking approaches require all DOs to communicate frequently which incurs an additional cost to the block generation process.

Table 7.1: Notation and terminology used in this chapter

B	Set of blocks of a database owner
BF	Inverted index of Bloom filters
CBT	Set of candidate block tuples
D	A database
MB	Set of mini-blocks
A	List of blocking attributes
B, B_i	A block and a block identifier
BC	Set of bit combinations
CBT	A candidate block tuple
Q	An queue data structure
$R, R.a, R.id$	A record, attribute value, and record identifier
R_G	A list of globally ranked bit positions
S	A set of q-grams
V_B, V_R	A binary and ratio vector
bf	A Bloom filter
b_{min}, b_{max}	Minimum and maximum block size
c, c_x, c_y	A centroid of a block
d	Number of database owners (or databases)
l_{bf}, l_{mhs}	Length of a Bloom filter, and Min-Hash signature
mb	A mini-block
mb_{min}, mb_{max}	Minimum and maximum mini-block size
$n_b, n_B, n_{CBT}, n_d, n_h, n_q, n_r$	Number of bits for splitting, bands, candidate block tuples, top-most dense bit positions, q-grams, and records
o_i	Number of 1's in bit position i
r_{seed}	Random seed value
$sim_C(), sim_J()$	Cosine and Jaccard similarity functions
bu	A hashed bucket in locality sensitive hashing
λ	A set of Min-Hash values
μ_i	An average value of bit position i
BF	Bloom filter
BK	Blocking key attribute
BKV	Blocking key value
BRP	Block representative pair
DO	A database owner
LSH	Locality sensitive hashing
PPRL	Privacy-preserving record linkage
MHS	Min-Hash signature
MD-PPRL	Multidatabase PPRL

To provide the DOs with control over the block generation process (in terms of the sizes and numbers of blocks) we propose a novel distributed blocking mechanism for MD-PPRL. Our proposed approach allows each DO to generate its set of blocks by clustering its own database independently without revealing any private information to any other DO.

Our proposed blocking approach follows the second linkage model described in Section 1.3 (on page 11) where we employ a LU to facilitate the identification of simi-

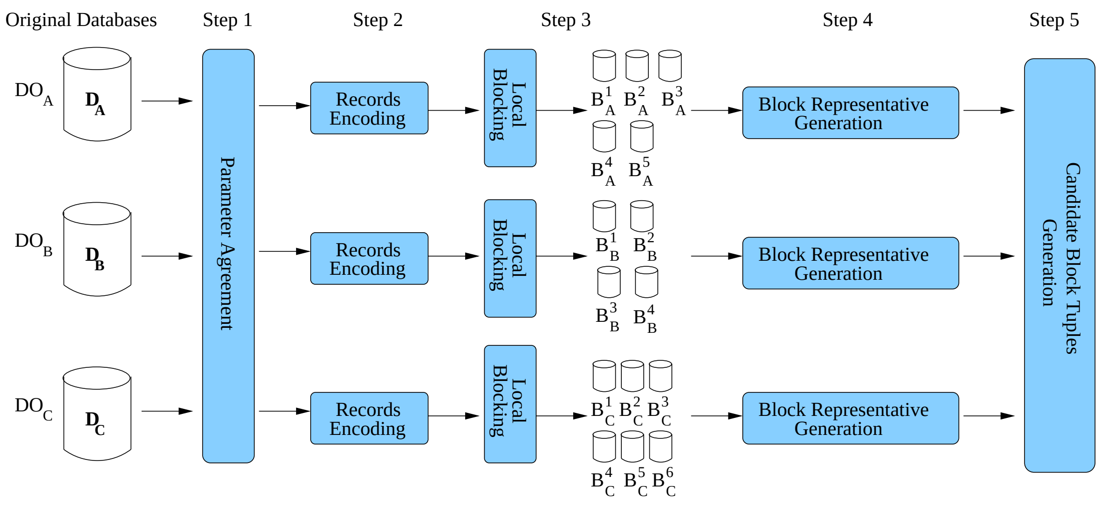


Figure 7.1: A high-level overview of our proposed approach for three database owners DO_A, DO_B, and DO_C, where $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}_C$ represent the databases of DO_A, DO_B, and DO_C, respectively. In step 1 all DOs need to agree on the parameter settings. In step 2 each DO independently encodes its database. In step 3 these encoded databases are blocked by the DOs independently as we describe in Section 7.3. Due to the independent nature of blocking each DO can have different number of blocks at the end of step 3. In step 4 each DO generates a set of block representatives for its generated blocks while in step 5 a list of candidate block tuples (CBTs) that need to be compared is generated based on these block representatives.

lar blocks for comparison. Once the blocks have been generated, each DO sends a set of representatives of its blocks to the LU. Then the LU applies a clustering technique on these sets of representatives to generate tuples of candidate blocks (described in more detail in Section 7.3 below) that need to be compared later in the privacy preserving record linkage (PPRL) process. Besides an initial exchange of parameter settings and the final central clustering step by the LU, our approach does not require any communication among the participating DOs. In Table 7.1 we summarise the notation we use in this chapter, and we next describe our approach in more detail.

7.2 Overview of our approach

As explained in Section 7.1 the main aim of our approach is to allow each DO to block their databases completely independently. Importantly, it allows DOs to block their databases according to their computational resources and privacy requirements (we will analyse privacy in detail in Section 7.4).

For ease of presentation we assume that the databases held by the DOs have the same database schema. If the databases have different schemas, a private schema matching approach can be used to align the different schemas [186]. Figure 7.1 illustrates a high-level overview of our approach.

As illustrated in Figure 7.1 our approach contains five main steps, which are:

1. *Parameter Agreement*: The DOs need to agree upon the parameter settings used in the record encoding and block representative generation process. The set of parameters will be described in more detail in Section 7.3.
2. *Record Encoding*: Each record in a database needs to be encoded before they are being grouped into blocks. It is important to note that the encoding technique depends on the blocking technique used in step 3 of our approach. However, for illustrative purposes we use q-grams and Bloom filters as described in Chapter 4 in the same way as in Chapters 5 and 6 as building blocks of the proposed approach for the encoding of records. We refer the reader to Section 4.3 for more details of each of these building blocks.
3. *Local Block Generation*: In this step each DO independently blocks its database. For the illustrative purpose we propose a clustering technique that can be used in this step where each DO performs clustering until the size of the generated blocks (clusters) falls within a specific lower and upper size range. In contrast to our previous blocking approaches (as detailed in Chapters 5 and 6) each DO can independently decide the size and number of blocks they want to generate.
4. *Block Representative Generation*: Once the blocks have been generated locally, each DO computes a lower dimensional representative for each of its blocks. Each DO then sends these representatives together with a block identifier to the LU. We will explain the generation of these representatives in more detail in Section 7.3.
5. *Candidate Block Tuple Generation*: To identify the blocks that need to be compared the LU applies a hashing based clustering technique on the received block representatives to generate candidate block tuples (CBTs).

Finally these merged blocks can be compared using private comparison and classification techniques to determine the matching record tuples in different databases [210]. We next describe each step of our approach in more details.

7.3 Distributed Scalable Blocking Approach

We assume d DOs are participating in our approach, each with a database and with $d \geq 3$. We assume a set A of quasi-identifiers [94], such as first name, last name, address details, etc., are common to all the databases. The attributes in A are used as blocking key (BK) attributes to group similar records into blocks. An illustrative example of our proposed approach is shown in Figure 7.2 and each step is discussed further below.

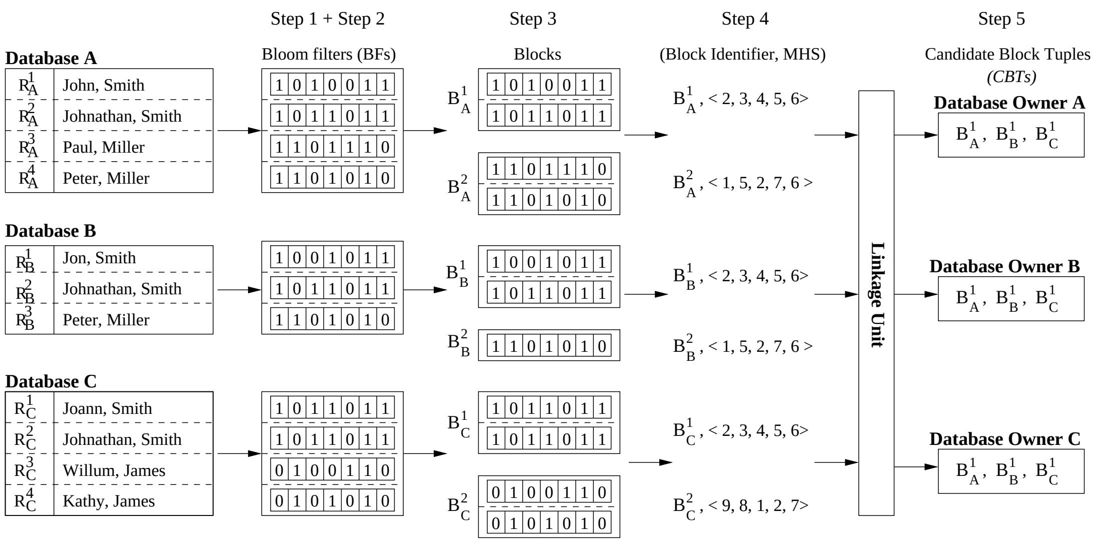


Figure 7.2: An illustrative example of candidate block tuple (CBTs) generation for three database owners DO_A, DO_B, and DO_C. In this example $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}_C$ represent the databases of DO_A, DO_B, and DO_C, respectively. In step 2 each DO uses Bloom filters (BFs) to encode its records. In step 3 the DOs block their databases independently, while in step 4 each DO sends a set of Min-Hash signatures (MHSs) as block representatives along with their identifiers to a linkage unit (LU). The LU applies a locality sensitive hashing technique on the received MHSs to generate candidate block tuples in step 5.

1. **Parameter Agreement**

   In step 1 of our approach, all DOs need to agree upon the set of parameters to be used for the record encoding and block representative generation. Since we are using Bloom filter (BF) encoding for illustrative purposes each DO needs to agree on the length of the BF, l_{bf}, the length (in characters) of q-grams, l_q, the number of hash functions, n_h, and the set A of BKs.

   To identify the similar blocks across different databases each DO needs to send a set of block representatives to the LU. We use Min-Hash signatures [25, 101] (MSH) as block representatives in our approach. To compute MSHs all DOs need to agree upon the number of top-most dense bit positions, n_d, Min-Hash signature length, l_{mhs}, and a random seed value, r_{seed}. We will describe each of these parameters in more details below.

2. **Record Encoding**

   As the second step of our approach, each DO independently encodes its database records into BFs as shown in Figure 7.2. We follow the same steps as in Algorithm 5.2 in Chapter 5, where each DO iterates over its database $\mathbf{D}$ and each record $R \in \mathbf{D}$ is encoded into a BF bf. Each bf is added to an inverted index data structure $\mathbf{BF}$ using its record identifier, $R.id$, as a key. Each DO then uses $\mathbf{BF}$ in the next step of the proposed approach to generate its set of blocks.

3. **Local Block Generation**

   As illustrated in Figure 7.2, the third step of our approach is to perform blocking over the BFs independently by each DO to generate a set of blocks. The distributed nature of our protocol allows each DO to select an appropriate blocking technique depending upon their computational resources. A key point is that the local block generation process can be considered as a *black box* where any blocking technique can be used same across the DOs as long as the minimum (b_{min}) and maximum (b_{max}) size of a block can be controlled. Each DO can set b_{min} and b_{max} independently without making any agreements with other DOs, which provides more control over the block generation process for the DOs. The output of this local block generation step is a set of blocks each containing a group of BFs.

   Clustering based Independent Blocking

   To allow each DO to independently block their databases we propose a novel threshold based hierarchical clustering technique based on the split and merge principle described in Chapter 6. Similar to the blocking technique proposed in Chapter 6, each DO first splits its inverted index **BF** into a set of *mini-blocks*.

Algorithm 7.1: Bloom filter splitting by a DO, adapted from Algorithm 6.1.

```
Input  : BF      - An inverted index of BFs
         $mb_{min}$  - Minimum size of a mini-block
         $mb_{max}$  - Maximum size of a mini-block
         $n_b$       - Maximum number of bit positions for splitting
         $R_G$       - Global ranking of bit positions
Output: MB       - Set of mini-blocks

1  $Q \leftarrow$[BF]                    // Initialisation of queue with the inverted index BF
2  while $Q \neq []$ do
3      $B \leftarrow Q.pop()$                       // Get the larget block at the front
4      $BC \leftarrow getCombinations(B, R_G, n_b)$           // Get bit combinations
5      $MB \leftarrow splitData(BC, B)$                      // Compute mini-blocks
6      if $\forall_{mb \in MB} |mb| \geq mb_{min}$ then    // Check if all mini-blocks are large than $mb_{min}$
7          foreach $mb \in MB$ do
8              if $|mb| \geq mb_{max}$ then          // Check if the mini-block is too large
9                  Q.push($mb$)                     // Add to queue for further splitting
10             end
11             else
12                 MB $\cup$ $mb$        // Add the mini-block to the final set of mini-blocks
13             end
14         end
15     end
16     else      // All bit combinations generate mini-blocks that are smaller than $mb_{min}$
17         MB $\cup$ $B$                     // Add current block to the set of mini-blocks
18     end
19 end
20 return MB
```

Database owner A (DO_A)

Bit pattern {**00**}

bf_A^5	**0**	1	**0**	0	1
bf_A^8	**0**	1	**0**	0	0

Bit pattern {**01**}

bf_A^1	**0**	1	**1**	0	0
bf_A^4	**0**	1	**1**	1	0
bf_A^7	**0**	1	**1**	1	0

Bit pattern {**10**}

bf_A^3	**1**	0	**0**	0	0
bf_A^6	**1**	1	**0**	1	0

Bit pattern {**11**}

bf_A^2	**1**	0	**1**	0	1

Database owner B (DO_B)

Bit pattern {**0**}

bf_B^1	1	0	**0**	1	1
bf_B^5	1	1	**0**	0	0
bf_B^7	0	0	**0**	1	1

Bit pattern {**1**}

bf_B^2	1	0	**1**	0	1
bf_B^3	0	0	**1**	1	0
bf_B^4	1	1	**1**	1	1
bf_B^6	0	0	**1**	0	1
bf_B^8	1	0	**1**	1	1

Database owner C (DO_C)

Bit pattern {**00**}

bf_C^2	**0**	1	**0**	1	0
bf_C^8	**0**	1	**0**	1	0

Bit pattern {**01**}

bf_C^1	**0**	1	**1**	1	0
bf_C^5	**0**	0	**1**	0	0
bf_C^7	**0**	1	**1**	0	1

Bit pattern {**10**}

bf_C^4	**1**	1	**0**	0	1
bf_C^6	**1**	0	**0**	1	0

Bit pattern {**11**}

bf_C^3	**1**	0	**1**	1	0

Figure 7.3: Mini-block generation according to a bit combination (by assuming DO_A, DO_B, and DO_C have set n_b as 2, 1, and 2, respectively). Assuming a global ranking of bit positions $R_G = \{3, 1, 4, 2, 5\}$, DO_A, DO_B, and DO_C select index positions {3,1}, {3}, and {3,1}, respectively, as splitting bit positions. According to the selected bit positions the respective **BF**s of DO_A, DO_B, and DO_C in Figure 5.3 (on page 96) are split into 4, 2, and 4 mini-blocks. The selected bit positions are shown in bold.

These generated mini-blocks are then merged until the size of a merged block reaches a minimum size, b_{min}, using a hierarchical merging approach. We next describe our clustering approach in more detail.

Splitting Phase

As the first step in our clustering technique, the BFs are split into sets of mini-blocks. Before starting the splitting process, the bit positions in BFs are ranked according to the ratios between the number of 0's and 1's for each bit position. We use Equation 6.1 on page 116 to calculate the 1/0 ratios of each bit position. We follow the same method described in Section 6.3 to rank the bit positions.

For example, the index positions of the globally summed ratio vector, V_G, in Figure 5.4 (on page 96), which is generated from inverted indexes of $\mathbf{BF}_A$, $\mathbf{BF}_B$, and $\mathbf{BF}_C$ of DO_A, DO_B, and DO_C, respectively, in Figure 5.3 (on page 96), can be ranked into a list of splitting bit positions, R_G, where $R_G = \{3, 1, 4, 2, 5\}$. The computed R_G is used to split the set of BFs of each DO iteratively until the BFs are grouped into smaller blocks within a specified size range, as outlined in Algorithm 7.1.

In contrast to Algorithm 6.1 on page 117, Algorithm 7.1 allows each DO to set parameters minimum (mb_{min}) and maximum (mb_{max}) mini-block size, and the number of splitting bit positions (n_b) independently. The parameters mb_{min}, mb_{max}, and n_b allow each DO to control the number of iterations that occur in the splitting algorithm as we will explain in more details in Section 7.4. As we have explained in Section 6.3, in each iteration the largest block of BFs that is available at the front of the queue Q is split into a set of mini-blocks by using n_b index positions in R_G as splitting bit positions (lines 3 to 5 in Algorithm 7.1).

Algorithm 7.2: Merging mini-blocks using hierarchical clustering

```
Input  : MB - Set of mini-blocks
         $b_{min}$ - Minimum size of a merged block
Output: B    - Set of merged blocks
B ← ∅                                   // Initialisation of the set of merged blocks
while MB ≠ ∅ do
    $mb_x$ ← MB.pop()                         // Get the frist mini-block
    $c_x$ ← computeCentroid($mb_x$)              // Compute the centroide
    S ← []                              // Initialise a list to keep similarities
    for $mb_y$ ∈ MB do                     // Iterate through remaining mini-blocks
        $c_y$ ← computeCentroid($mb_y$)          // Compute the centroide
        s ← $sim_C(c_x, c_y)$               // Compute the similarity between the centroides
        S.add(s)                        // Add the similarity to the similarity list
    end
    $mb_y$ ← getMinBlock(S,MB)           // Get the mini-block with the highest similarity
    B ← $mb_x \cup mb_y$                    // Merged the mini-blocks
    MB.delete($mb_x$,$mb_y$)                 // Delete the merged mini-blocks
    if $|B| < b_{min}$ then              // Check if the merged block is not large enough
        MB.add(B)                       // Add the merged block back to set of mini-blocks
    end
    else
        B.add(B)                        // Add the merged block to the set of merged blocks
    end
end
return B
```

If all of the resulting mini-blocks contain BFs greater than mb_{min} (line 6), then each $mb \in MB$ is checked against the value of mb_{max} (lines 7 to 9). Similar to Algorithm 6.1, any mini-block that is larger than mb_{max} is added to Q for future splitting (line 9), while others are added to the set of mini-blocks **MB** (line 12). In the next phase the mini-blocks in **MB** are merged until the resulting blocks contain a minimum number of BFs. We refer the reader to Section 6.3 on page 115 for more details on the mini-block generation process. Figure 7.3 provides an example of generating *mini-blocks* for the sets of BFs in Figure 5.3.

Merging Phase

For merging of mini-blocks we suggest a threshold based hierarchical clustering algorithm which guarantees that mini-blocks are only merged upto the minimum size limit b_{min}, as outlined in Algorithm 7.2. As explained in Section 6.3 limiting the minimum block size allows us to perform computationally efficient blocking. Before starting the clustering algorithm, each DO can independently set an appropriate value for b_{min} which controls the size of its merged blocks. However, due to the iterative merging of mini-blocks, the parameter b_{max} can only be within the size range of $b_{min} \leq b_{max} \leq 2(b_{min} - 1)$ because each merged block contains at least b_{min} and at most $2(b_{min} - 1)$ BFs where latter occurs when two mini-blocks of size $(b_{min} - 1)$ are merged.

As detailed in Algorithm 7.2, to merge mini-blocks each DO iterates over its set of mini-blocks **MB** (line 2). At each iteration one mini-block mb_x is selected and the respective centroid c_x is computed (lines 3 and 4). The function *compute-Centroid()* computes centroid $c = [\mu_1, \mu_2, \mu_3, \cdots, \mu_{l_{bf}}]$ as a ratio vector for each mini-block, where μ_i is the average number of BFs that have a 1 bit at position i and l_{bf} is the length of a BF. For a given mini-block mb, μ_i can be calculated as $\mu_i = \frac{o_i}{|mb|}, 1 \leq i \leq l_{bf}$, where o_i is number BFs that have a 1-bit at position i and $|mb|$ is the number of BFs in mb. We use the Cosine similarity (sim_C) between centroids for computing the similarity of mini-blocks because this similarity function is commonly used in high dimensional similarity calculations [229]. sim_C between two centroids c_x and c_y is calculated using Equation 7.1:

$$sim_C(c_x, c_y) = \frac{c_x \cdot c_y}{||c_x||||c_y||} = \frac{\sum_{i=1}^{l_{bf}} c_x^i \times c_y^i}{\sqrt{\sum_{i=1}^{l_{bf}} (c_x^i)^2} \times \sqrt{\sum_{i=1}^{l_{bf}} (c_y^i)^2}} \quad (7.1)$$

This similarity is computed between the initial mini-block mb_x and every other mini-block mb_y in **MB** (lines 6 to 10). The mini-block mb_y that has the highest similarity with mb_x is merged with mb_x into the new block of B (lines 11 and 12). The merged mini-blocks mb_x and mb_y are removed from **MB** to avoid repetitive merging with other mini-blocks (line 13).

After the selected two mini-blocks have been merged the size of the resulting block B is checked against the minimum block size b_{min} (line 14). If the size of B is less than b_{min}, B is added back into **MB** (line 15). This enables B to be merged further with other similar mini-blocks. Once the size of B reaches at least b_{min}, B is added to the final set of merged blocks **B** (line 18).

However, a DO could use the canopy clustering techniques as described in Section 6.3 for the merging of mini-blocks. To do so each DO needs to select their own similarity (loose and tight) thresholds and follow the steps outlined in Algorithms 6.2 and 6.3. Hence, a BF needs to be computed as the centroid for each mini-block, and mini-blocks are merged based on the Hamming similarity as calculated in Equation 6.4.

4. **Block Representative Generation**

As illustrated in Figure 7.1, in the fourth step in our approach each DO generates a block representative pair (BRP) for each merged block B in **B**. A BRP consists of a block identifier (B_i) and a Min-Hash signature (MHS) [25] in the form of $\langle B_i$, MHS$\rangle$.

Prior to the BRP generation, for each block $B \in$ **B** a ratio vector V_R is computed that defines the densities of all l_{bf} bit positions of the BFs in a cluster, which allows each DO to select the most dense bit positions in a block for the signature generation, and hide information about less dense bit positions which can potentially reveal sensitive information to other DOs [56, 132]. The density of a bit position i is calculated as $\mu_i = \frac{o_i}{|B|}, 1 \leq i \leq l_{bf}$, where o_i is the number of 1-bits in position i and $|B|$ is the number of BFs in a given block B. Then, the bit positions in V_R are ranked according to the density values in descending order and the $n_d (< l_{bf})$ top-most dense bit positions are selected. Each V_R is then converted into a bit vector V_B, where all the selected n_d bits are set to 1 and the other $(l_{bf} - n_d)$ positions are set to 0. Finally, a V_B is used to generate the MHS for its block B.

For example, assume a $V_R = [0.125, 0.1, 0.01, 0.5, 0.25]$ for a given block B. Suppose we set the $n_d = 3$ which allows to select the top three dense bit positions at most. According to the ranking order the bit positions selected are $(1, 4, 5)$. Thus, the resulting bit vector V_B for V_R of B is $< 1, 0, 0, 1, 1 >$.

Min-Hash is a class of hash functions that approximate the Jaccard similarity [25]. The idea behind using Min-Hash is that the probability of two binary vectors V_B and V'_B to generate the same Min-Hash value equals the Jaccard similarity of those two vectors. Formally, the probability that V_B and V'_B will generate the same min-hash for a hash function g_i can be defined as follows [25]:

$$Pr[g_i(V_B) = g_i(V'_B)] = \frac{|V_B \cap V'_B|}{|V_B \cup V'_B|} = sim_J(V_B, V'_B), \tag{7.2}$$

where $|V_B \cap V'_B|$ is the number of 1-bits common in V_B and V'_B, $|V_B \cup V'_B|$ is the number 1-bits appearing in both V_B and V'_B, and $sim_J()$ is the Jaccard similarity between V_B and V'_B. Hence, the probability Pr increases with the Jaccard similarity between the two vectors.

To compute a MHS, we select l_{mhs} independent hash values for each index in V_B by using a random seed value r_{seed}, which defines l_{mhs} random permutations [25]. Using a single pass over the set of V_B, the corresponding n_l Min-Hash values $g_1(V_B)$, $g_2(V_B)$, $\cdots$, $g_{l_{mhs}}(V_B)$ are determined for each V_B under each of the l_{mhs} random permutations. Note that each $g_i()$ returns the index (i.e., position) in which the first 1-bit occurs in V_B which becomes ith of component of a MHS. Finally, the generated MHS is used in the BRP for each block.

As an example of generating a MHS, let us assume a binary vector $V_B = [1, 0, 1, 1, 0]$. Let us create the MHS of $V_B = \{g_1, g_2\}$. Suppose we select two random permutations $RP_1 = (2, 3, 5, 4, 1)$ and $RP_2 = (5, 4, 1, 3, 2)$ for the two hash functions g_1 and g_2, respectively. By applying the RP_1 to V_B we get the permuted V_B as $RP_1([1, 0, 1, 1, 0]) = [0, 1, 0, 1, 1]$. To compute the Min-Hash value of $g_1(V_B)$ we then apply the hash function g_1 to the permuted V_B which gives the Min-Hash value of $g_1([0, 1, 0, 1, 1]) = 3$. Similarly, for hash function g_2 we get the Min-Hash value $g_2(V_B) = RP_2([1, 0, 1, 1, 0]) = g_2([0, 1, 1, 1, 0]) = 4$. Hence, the MHS of $V_B = \{g_1, g_2\} = \{3, 4\}$. Before generating the BRPs independently, all DOs need to agree on the parameters n_d, l_{mhs}, and r_{seed}, and the importance of each is described next.

Number of top-most dense bit positions (n_d): All DOs need to use the same number of top most dense bit positions, since a different number of dense bit positions can generate different MHS for similar blocks at different DOs. This would result in these blocks not being considered as candidates for comparison in step 4 of our protocol. However, before agreeing on a common n_d value the DOs can calculate an interval of the minimum and maximum number of dense bit positions that can be selected [212]. The maximum number of dense bit positions is important in terms of privacy because less dense bit positions could potentially provide frequency information about rare q-grams. On the other hand, the minimum number of dense bit positions provides a limitation on the overlap between the MSHs of clusters because the use of only the few most dense bit positions can generate the same Min-Hash values for all clusters which potentially end up with the same MSH for all clusters. Therefore, each DO could share its interval n_d value ranges with other DOs and select a suitable value for n_d that is within all these intervals. We leave the development of a technique to calculate an optimal value for n_d as a future work.

The length of a Min-Hash signature (l_{mhs}): The length l_{mhs} provides a trade-off between accuracy and computational cost in step 5 of our approach. In general, the probability that two bit vectors V_B and V'_B generate the same Min-Hash value for a hash function g_i is computed using Equation 7.2.

For MHSs of length l_{mhs}, the $sim_J(V_B, V'_B)$ can be estimated as,

$$sim_J(V_B, V'_B) \approx y/l_{mhs} = |\{i|1 \leq i \leq l_{mhs} \text{ } and \text{ } g_i(V_B) = g_i(V'_B)\}|/l_{mhs}, \quad (7.3)$$

where y is the number of hash functions for which $g_i(V_B) = g_i(V'_B)$. If $sim_J(V_B, V'_B)$ deviates from y/l_{mhs} with an error ϵ, then the probability that $y = (1-\epsilon) \cdot sim_J(V_B, V'_B) \cdot l_{mhs}$ Min-Hash values of V_B and V'_B are equal can be defined as:

$$Pr[y, l, sim_J(V_B, V'_B)] = \sum_{j=0}^{y} \binom{l}{y} sim_J(V_B, V'_B)^j \cdot (1 - sim_J(V_B, V'_B))^{l-j} \quad (7.4)$$

By applying the Chernoff Bound [153] to Equation 7.4, the probability that $sim_J(V_B,V'_B)$ deviates from y/l_{mhs} up to an error bound ϵ is given by:

$$Pr[|sim_J(V_B, V'_B) - (y/l_{mhs})| \leq \epsilon] \leq 2e^{-\frac{\epsilon^2}{2+\epsilon} l_{mhs}} < \delta, \quad (7.5)$$

where $0 < \epsilon \leq 1$. Equation 7.5 must satisfy $l \geq (2+\epsilon)/\epsilon^2 \cdot ln(2/\delta) = O(1/\epsilon^2)$ with a probability $\delta > 0$ [46]. To this end, for any constant $l_{mhs} = O(1/\epsilon^2)$ the expected error of the similarity estimate is at most ϵ. Based on estimation of ϵ the DOs can agree upon an appropriate value for l_{mhs} (with $l_{mhs} < l_{bf}$).

Random seed value for permutation generation (r_{seed}): The value r_{seed} needs to be shared among the DOs to generate the same set of l_{mhs} permutations for the MHS generation using a pseudo-random generator.

Algorithm 7.3: Generating candidate block tuples using Locality Sensitive Hashing (LSH) by the LU

```
Input  : BRP - Sets of BRPs from DOs
         $n_B$ - Number of bands considered in a MHS
         $l_{mhs}$ - Length of a MHS
         $n_{CBT}$ - Number of candidate block tuples need to be matched
Output: CBT - Set of candidate block tuples
$MHS_B \leftarrow []$; CBT $\leftarrow \emptyset$; $CT \leftarrow \{\}$          // Initialisation of variables
$\lambda \leftarrow l_{mhs}/n_B$                              // Calculate the number of hash values in a band
foreach $i \in 1,2,\ldots,n_B$ do                   // Each MHS is hashed for all the bands
    $MHS_B[i] \leftarrow \{\}$                          // Create an inverted index for band $i$
    foreach $j \in 1,2,\ldots,d$ do                      // Hash MHSs from all the DOs
        foreach $\langle B_{id}, B_{MHS}\rangle \in$ BRP$[j]$ do
            $bu \leftarrow getMinHashValues(B_{MHS}, i, \lambda)$     // Get corresponding Min-Hashes
            if $bu \in MHS_B[i]$ then   // Check if a bucket is available with hash key $bu$
                $MHS_B[i][bu].add(B_{id})$                   // Add the $B_{id}$ to the bucket
            end
            else
                $MHS_B[i][bu] \leftarrow [B_{id}]$     // Create a new bucket with $\lambda$ Min-Hash values
            end
        end
    end
    foreach $bu \in MHS_B[i]$ do                        // Iterate over each bucket
        $C \leftarrow createCandidateTuples(MHS_B[i][bu])$    // Compute candidate block tuples
        foreach $CBT \in C$ do
            $CT[CBT] \leftarrow incrementCounter(CT, CBT)$  // Increment the band counter by 1
        end
    end
end
CBT $\leftarrow getCandidateTuples(CT, n_{CBT})$              // Get $n_{CBT}$ candidate block tuples
return CBT
```

5. **Candidate Block Tuple Generation**

   In the last step of our approach, each DO sends its generated BRPs to the LU as illustrated in Figure 7.1. In order to identify the candidate block tuples (CBTs), as a naive approach the LU can compute the similarities between all MHSs and based on these similarity values group the BRPs into different CBTs. This would require a complexity of $O(|\mathbf{B}|^d)$, if each of the d DOs generates $|\mathbf{B}|$ blocks, which can become infeasible in terms of the number of computations. To improve the efficiency of generating CBTs, a Locality Sensitive Hashing (LSH) approach is used, as outlined in Algorithm 7.3. LSH is commonly used for searching nearest neighbours in high dimensional data [101]. LSH has been used in a PPRL context where Durham [56] proposed Hamming distance based LSH functions to efficiently generate candidate record pairs, while Karapiperis and Verykios [118] have suggested a two database PPRL blocking framework which also uses LSH functions for blocking.

LSH uses a set of hash functions to map objects into partitions in such a way that similar objects are mapped to the same partition (bucket) with high probability, while guaranteeing that dissimilar objects are not mapped into the same bucket with high probability [101]. In our approach the LU applies LSH over the MHSs sent by the DOs to group these MHSs into different buckets with the aim that dissimilar MHSs will never hash to the same bucket while similar MHSs will hash to the same bucket under at least one of the hash values in these signatures.

LSH involves the use of a family of hash functions, which in our approach can be defined as follows. The MHSs of each DO are divided into n_B bands each consisting of λ Min-Hash values where $l_{mhs} = n_B \cdot \lambda$. If two signatures, MHS$_1$ and MHS$_2$, have a Jaccard similarity s, then the probability that the signatures agree in all λ Min-Hash values of one particular band is s^λ [101]. Thus, the probability that MHS$_1$ and MHS$_2$ agree in all λ hash values of at least one band to become a candidate pair is $1-(1-s^\lambda)^{n_B}$ (we refer the reader to [25] for more details). This defines the LSH family to be $(s_1, s_2, 1-(1-s_1^\lambda)^{n_B}, 1-(1-s_2^\lambda)^{n_B})$-sensitive if,

- sim_J(MHS$_1$,MHS$_2$)$\geq s_1$, then $Pr_{i\in 1,\ldots,n_B}[\{ba_i|\text{MHS}_1\} = \{ba_i|\text{MHS}_2\}] \geq p_1$, and
- sim_J(MHS$_1$,MHS$_2$)$\leq s_2$, then $Pr_{i\in 1,\ldots,n_B}[\{ba_i|\text{MHS}_1\} = \{ba_i|\text{MHS}_2\}] \leq p_2$,

where p_1 and p_2 are two probabilities such that $p_1 > p_2$, ba_i is a band in a MHS, and s_1, s_2 (with $1 \geq s_1 > s_2 \geq 0$) are two Jaccard similarity thresholds.

As outlined in Algorithm 7.3, the MHSs of all DOs are hashed for each ith band into a list of inverted indexes MHS_B (lines 2 to 10). For a given ith band ba_i, the Min-Hash values of a MHS are used as the hashing key of a bucket $bu \in MHS_B[i]$ by concatenating all Min-Hash values and the corresponding B_{id} is added to bu (lines 7 to 10). If MHSs of different DOs have the same λ values for band ba_i then the corresponding B_{id}s are added to the same bucket in $MHS_B[i]$ which become a CBT to be compared.

The LU then generates CBTs using the function *createCandidateTuples()* which provides the set of CBTs that need to be compared among all the DOs (line 17). For example if a bucket u contains B_{id}s B_A^1 and B_A^2 of DO$_A$, B_B^1 of DO$_B$, and B_C^1 of DO$_C$ then the function *createCandidateTuples()* returns a list $C = [\langle B_A^1, B_B^1, B_C^1\rangle, \langle B_A^2, B_B^1, B_C^1\rangle]$ containing two CBTs of size three. Each CBT that is generated for a given bucket bu is added to an inverted index CT with a counter value, defining the number of times a given CBT is generated for n_B bands (lines 18 to 20). A CBT gets a higher counter value if its respective MHSs are more similar compared to a CBT with its respective MHSs are dissimilar.

Rather than comparing each CBT in CT, the DOs can agree upon the number of CBTs that are to be compared in the comparison step later in the PPRL pipeline. The parameter n_{CBT} specifies the maximum number of CBTs that are to be generated by the LU, and is computed based on an approximation of the reduction ratio (RR). By assuming each DO generates $|\mathbf{B}|$ blocks (each of the same size), then the total number of block comparisons for d DOs is equal to $n_T = |B|^d$. The approximate RR can be considered as the fraction of block comparisons reduced from n_T, computed as $RR_{approx} \approx 1 - (n_{CBT}/n_T)$. All DOs need to agree on a suitable value for RR_{approx} to calculate how many block comparisons they want to perform in the comparison and classification step later in the PPRL pipeline.

The selection of n_{CBT} CBTs allows the LU to identify the block tuples which are more likely to be similar among all the CBTs generated. To identify the required CBTs that need to be compared, first the CBTs are ranked in a descending order based on the respective counter values in CT which defines how often a given CBT is generated for a bucket $bu \in MHS_B$. A CBT generated for all $bu \in MHS_B$ gets the highest rank. The top n_{CBT} CBTs are returned from the function *getCandidateTuples()* and are added to the set **CBT** (line 23). Finally, **CBT** is sent to all DOs as shown in Figure 7.1.

7.4 Conceptual Analysis of Distributed Blocking Scheme

In this section we analyse our private blocking approach in terms of complexity, privacy, and quality of blocking.

7.4.1 Complexity

Let us assume $d \geq 3$ database owners (DOs) are participating in our approach with each having a database $\mathbf{D}$ to be blocked. By assuming there are n_r records in $\mathbf{D}$ with each record R having an average of n_q q-grams in blocking attributes A, we analyse the computational and communication complexities in terms of a single DO.

In step 2 of our approach we assume each DO encodes all records in its database $\mathbf{D}$ into Bloom filters (BFs) of length l_{bf} using n_h hash functions independently. The record encoding step is of $O(n_h \cdot n_q \cdot n_r)$ complexity since the BF encoding is applied to all records in a linear manner.

In step 3, each DO needs to block its generated BFs. Let as assume the proposed clustering based independent blocking approach is used in this step. In the splitting phase of our proposed hierarchical clustering technique the parameters mb_{min} and mb_{max} are used to control the size of mini-blocks that are generated. In each splitting iteration a given block B is split into $|MB| = 2^{|BC|}$ mini-blocks according to a selected bit combination BC. Therefore, $log_{|MB|}(\frac{n_r}{mb_{max}})$ splitting iterations are required to split n_r records. Hence, the splitting of n_r BFs into a set of mini-blocks is of $O(n_r \cdot log_{|MB|}(\frac{n_r}{mb_{max}}))$.

In the merging phase of our clustering technique the mini-blocks are merged based on the similarities of centroids in each merging iteration. The computation of a centroid for a given mini-block mb requires to generate a ratio vector V_R. The generation of each V_R is of $O(l_{bf} \cdot |mb|)$ complexity. In each merging iteration the two most similar mini-blocks are merged together. Therefore, the merging phase of our clustering technique requires a total of $O((\frac{n_r}{mb_{max}})^2)$ computations.

In step 4 we assume each DO generates $\frac{n_r}{b_{min}}$ blocks. To compute MHSs as block representatives, each DO calculates a V_R for each of its generated blocks. This requires a complexity of $O(l_{bf} \cdot \frac{n_r}{b_{min}})$, where b_{min} defines the minimum number of BFs in a block. Consequently, generating l_{mhs} length MHSs for all blocks has a complexity of $O(l_{mhs} \cdot \frac{n_r}{b_{min}})$. In step 5 the LU needs to hash MHSs of all the d DOs into n_B bands which has a complexity of $O(n_B \cdot d \cdot \frac{n_r}{b_{min}})$. For each bucket in MHS_B the function *createCandidateTuples()* returns a list of candidate block tuples which requires a total computational complexity of $O(n_B \cdot \sum_{i=1}^{n_B} |MHS_B[i]|)$.

By excluding the initial parameter agreements in our approach, which has a constant complexity, the DOs communicate with the LU only in step 4 where each DO sends its generated set of BRPs to the LU. The LU receives d messages each containing $\frac{n_r}{b_{min}}$ BRPs which leads to an overall communication complexity of $O(l_{mhs} \cdot d \cdot \frac{n_r}{b_{min}})$.

7.4.2 Blocking Quality

We analyse the quality of our protocol in terms of effectiveness and efficiency [35]. Since each DO performs local blocking independently in step 3 of our approach, the overall blocking quality depends upon the blocking technique used by the DOs.

The lower (mb_{min}) and upper (mb_{max}) size bound of the clusters generated by each party determine the number of mini-blocks generated in the splitting step in the proposed hierarchical clustering technique. As we have discussed in Section 7.3, splitting sets of BFs into smaller mini-blocks and merging similar mini-blocks upto b_{min} ensures similar records are grouped into blocks which improves the overall quality of our approach. The parameter n_b needs to be selected appropriately since the process of generating bit combinations grows exponentially with n_b which can increase the overall runtime of our approach. However, as we have seen in Section 6.5, an increase of n_b improves the quality of blocking since more similar BFs are grouped into mini-blocks which improves the effectiveness of merging in clustering.

In step 4 of our approach, the length l_{mhs} of a MHS provides a trade-off between effectiveness and efficiency of the candidate block tuple (CBT) generation in step 5. An increase of l_{mhs} decreases the error ϵ of the similarity estimation by $\epsilon = O(1/\sqrt{l_{mhs}})$, which would decrease the *false negative* rate [36], but would increase the number of *false positives* [36] in the CBT generation.

For a given signature length l_{mhs}, the runtime and quality of the CBT generation in step 5 depends upon the number of bands n_B considered by the LU. As n_B decreases the probability that MHSs are mapped to the same bucket decreases, which leads to the number of false positives to decrease, but it would also increase the number of false negatives. On the other hand, incrementing n_B requires more computational time as each MHS is hashed more often which leads to an increase in the number of CBTs. To assess the blocking quality we evaluate our approach in the next section with different large datasets under different parameter settings.

7.4.3 Privacy

We assume each DO follows the honest-but-curious (HBC) adversary model (as described in Chapter 2). In steps 1 to 4 of our approach, each DO performs its computations independently without any communication across the DOs except for the parameter agreements, which does not reveal any private information about the databases held by the DOs. In step 3 of our approach each DO selects the values of mb_{min}, mb_{max}, and b_{min} according to their privacy requirements. This improves the overall privacy of our approach since the number of records in blocks of a given DO is not known to any other party participating in our approach. Also, the parameter b_{min} guarantees that every merged block of a DO contains at least b_{min} records, which guarantees k-anonymous mapping ($k = b_{min}$) privacy [212, 216].

The LU is not capable of deducing anything about the DO's databases as the LU only receives a set of BRPs from each DO. The parameters used in the BF generation, the values for n_d, r_{seed}, and the sizes of the blocks, all are unknown to the LU. Therefore, the LU cannot learn the frequency distribution of the blocks generated to conduct a frequency attack [212].

However, under the HBC model collusion is possible [140, 212]. In our approach one or more DOs can collude with the LU to deduce information about other DOs. A colluding DO can reveal the set of parameters used in the BF and MHS generation processes to the LU. However, our approach guarantees the privacy of a database of a DO in two different ways. Firstly, the local block generation allows each DO to block their database independently without sharing its blocking parameters with any other DO. This ensures that even if a DO reveals its encoding parameters to the LU, the LU still does not know block sizes to conduct a frequency analysis. Secondly, the generated blocks are not being sent to the LU except for a set of representatives. In the signature generation process the parameter top most dense bit position (n_d) ensures the LU cannot learn any frequency information about the rare q-grams assigned to less dense bit positions. Therefore, the LU cannot exactly match a record to a MHS even if it would regenerate the respective binary vectors of MHSs that it received from a non-colluding DO.

7.5 Experimental Evaluation

In this section, we present the experimental evaluation of our distributed blocking scheme. Table 7.2 summarises the datasets and parameter values used. We use the clustering approach proposed in Section 7.3 in the local block generation step of our approach.

Table 7.2: Parameter settings used in the experimental evaluation

Parameter	Value range
Dataset name	NC-SYN
Number of databases (d)	3, 5, 7, 10
Number of records in each database (n_R)	5,000 to 1,000,000
Blocking key attributes (A)	Given name, Surname, City (suburb), and Postcode
Corruption levels	0%, 20%, and 40%
Bloom filter length (l_{bf})	1000
Number of hash functions (n_h)	30
Character length of q-gram (n_q)	2
Minimum mini-block size (mb_{min})	10
Maximum mini-block size (mb_{max})	20
Number of splitting bit positions (n_b)	3
Minimum merged block size (b_{min})	50, 100, 500, 1000
Number of top-most dense bit positions (n_d)	500
Length of a Min-Hash signature (l_{mhs})	250, 500, 1000
Number of Min-Hash values in a band (λ)	2, 5, 10

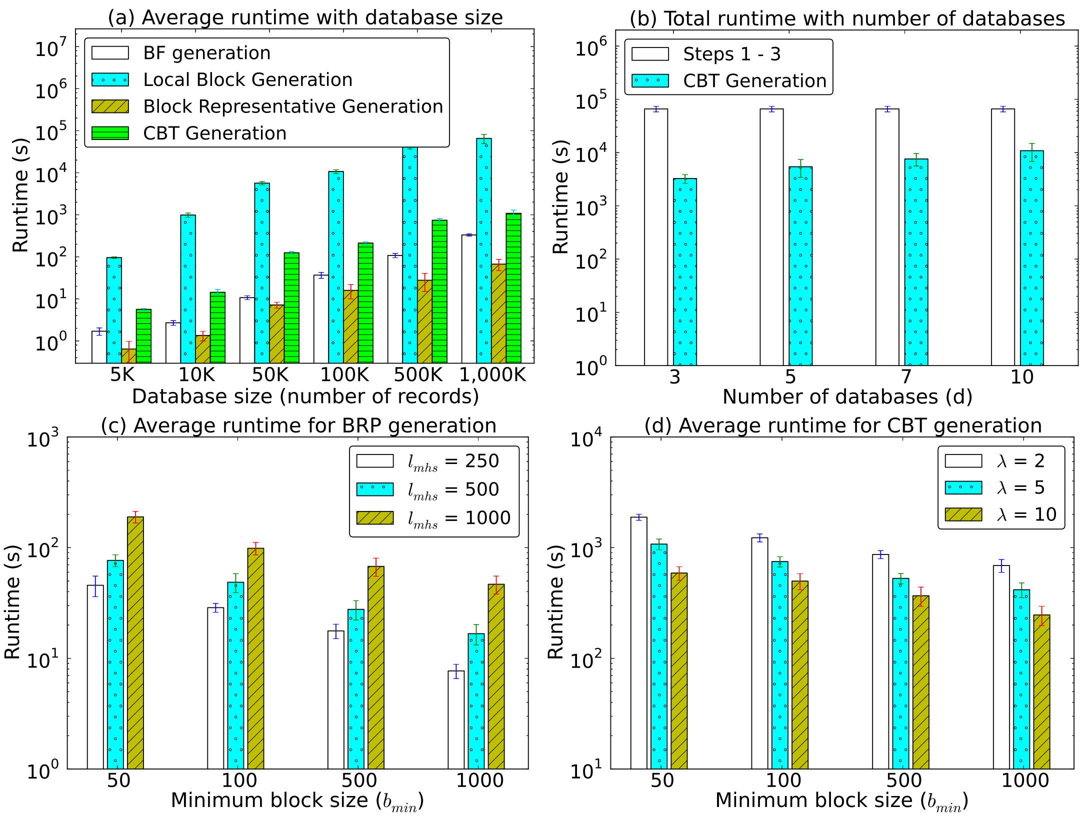


Figure 7.4: (a) Average runtime for each step of our distributed blocking scheme with database size, (b) the total runtime with different number of databases, and average runtime for (c) block representative pair (BRP) generation and (d) the candidate block tuples (CBTs) generation with minimum block size (b_{min}). Note that plots have different y-axis scales.

7.5.1 Scalability

Similar to Chapters 5 and 6, we measured the average and total runtime of our distributed blocking approach to evaluate the complexity. Figure 7.4 (a) shows the scalability of each step of our approach in terms of database sizes. As expected the runtime required for Bloom filter (BF) generation increases linearly with database size as more time is required to encode more records into BFs. As can be seen in this figure, the local block generation step uses more runtime to generate blocks when the database size is increasing because of the hierarchical merging process in the clustering technique used. However, the efficiency of local block generation can be improved by using a blocking technique proposed in Chapters 5 or 6.

As can be seen in Figure 7.4 (a), the runtime required for block representative pair (BRP) generation increases linearly with database size because more blocks are generated for a given minimum block size (b_{min}) when database size is increasing. The average runtime requires for CBT generation increases linearly with database size because more BRPs are hashed into different buckets in the LSH process.

Figure 7.4 (b) shows the total runtime required for steps 1 to 4 of our approach with different numbers of databases (d). For steps 1 to 3 our approach requires the same runtime for different number of databases because each DO performs these steps independently (assuming all DOs perform step 1 to 3 at the same time). However, this figure shows that the runtime required for CBT generation increases with d because more hashing is performed by the LU in the LSH process.

Figures 7.4 (c) and (d) shows the average runtime required for the BRP and CBT generation steps with different minimum block sizes (b_{min}) for a database with 1 million records. As can be seen in Figure 7.4 (c), the average runtime for BRP generation increases linearly with the Min-Hash signature length (l_{mhs}) because more random permutations are used to generate the required number of Min-Hash values. Figure 7.4 (d) shows the average runtime for CBT generation decreases with the band size (λ) used in the LSH process. This is because the number of buckets generated in LSH decreases when λ increases that reduces the overall runtime required in the CBT generation step. However, these figures show the average runtime required for BRP and CBT generation steps decreases as b_{min} increases. This is because a smaller number of blocks are generated per database when b_{min} increases.

7.5.2 Blocking Quality

We measured the blocking quality of our approach in terms of pairs completeness (PC) for approximated reduction ratio (RR) calculated as described in Section 7.3. Figure 7.5 illustrates PC against RR for a database with 1 million records for different number of databases and different corruption levels (0%, 20%, 40%) when all databases have been blocked using the same block size. According to the results illustrated in this figure, our approach achieves high PC rates even in the presence of dirty data in these databases. We noted that PC is increasing with the number of databases with low quality data (20% and 40% corruption levels) for a given RR value. This is because more CBTs are compared for a larger number of databases

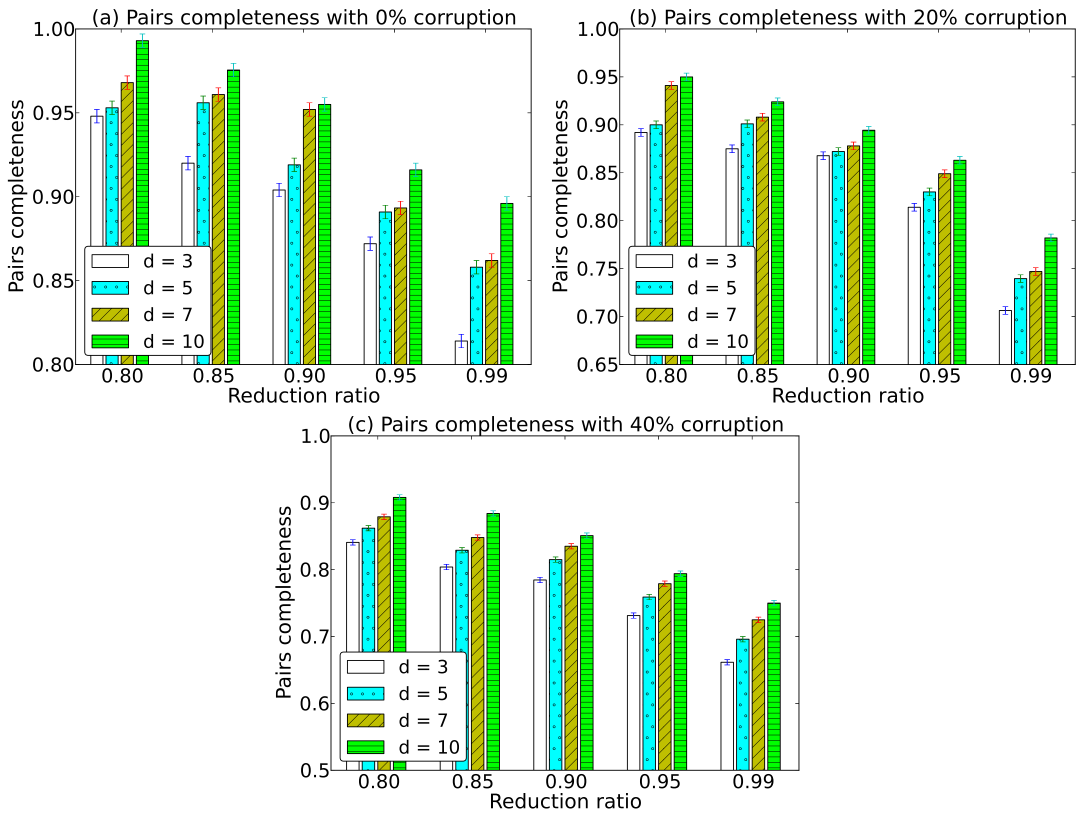


Figure 7.5: Pairs completeness (PC) measured against reduction ratio (RR) with different number of databases for (a) 0%, (b) 20%, and (c) 40% corruption levels for databases with 1,000,000 records. Note that plots have different y-axis scales.

based on the computed RR value. As a result, with the increment in the number of databases that need to be blocked, a lower RR value would require more candidate record tuples (*CRT*s) to be compared in the private comparison and classification step in PPRL.

Figure 7.6 shows PC for blocking of three databases with the use of different Min-Hash signature lengths (l_{mhs}) and band sizes (λ) in the generation of BDPs and *CBT*s, respectively. As can be seen in Figure 7.6 (a), PC increases with l_{mhs} because more similar blocks are hashed into the same bucket in the LSH process during the *CBT* generation. However, PC decreases with λ because similar blocks might not be hashed to the same bucket based on the selected Min-Hash values. Therefore, the use of larger Min-Hash signatures with a small band size in the *CBT* generation improves blocking quality of our approach, however, it will also increase the overall runtime as seen in Figures 7.4 (c) and (d). Figure 7.6 (d) illustrates PC against RR for blocking of three databases with 1 million of records with different block sizes, which shows our approach provides high blocking quality for lower block sizes and even when the DOs used different block sizes.

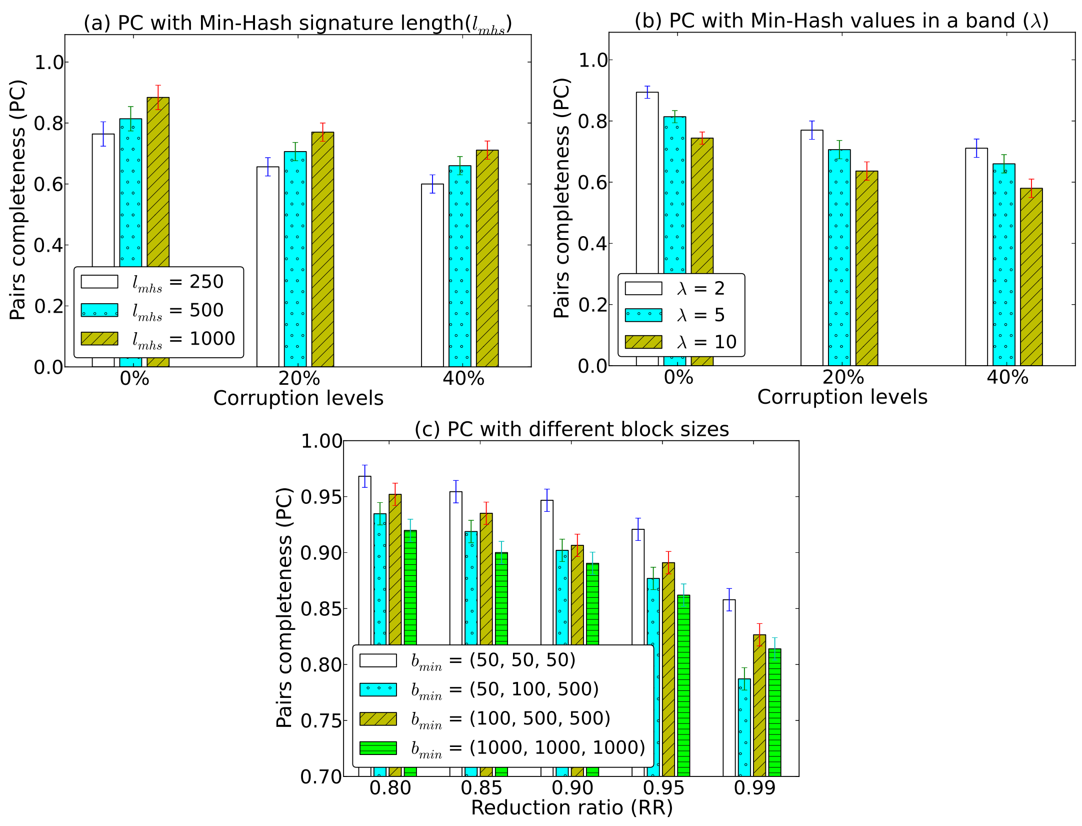


Figure 7.6: Pairs completeness (PC) with different (a) Min-Hash signature length (l_{mhs}), (b) band sizes (λ), and (c) block sizes (b_{min}) for blocking of 3 databases. Note that plots have different y-axis scales.

7.5.3 Privacy

As shown in Figure 7.7 (a), we measured the average (DS_{Mean}), median (DS_{Med}), and maximum (DS_{Max}) disclosure risk values with different b_{min} values (as we described in Section 4.4.2). As shown in this figure, the DR values decrease for large block sizes as more records are included in a given block making our blocking approach more secure against frequency attacks.

We also studied the sizes of blocks generated for different minimum block size (b_{min}) values, as shown in Figure 7.7 (b). This figure illustrates that our proposed clustering approach generates blocks that contain at least b_{min} records ensuring k-anonymous privacy where k =b_{min}. We also noted that the variance between block sizes generated by the clustering approach gets lower when b_{min} increases. This suggests that the generation of blocks independently provides more control for the DOs over their block sizes. Similar to the clustering approaches proposed in Chapter 6, we noted that for smaller b_{min} (when b_{min} = 50) the hierarchical clustering could result in blocks with a large number of records (> b_{min}) due to the iterative splitting used. However, the number of such large blocks decreases when b_{min} is increased.

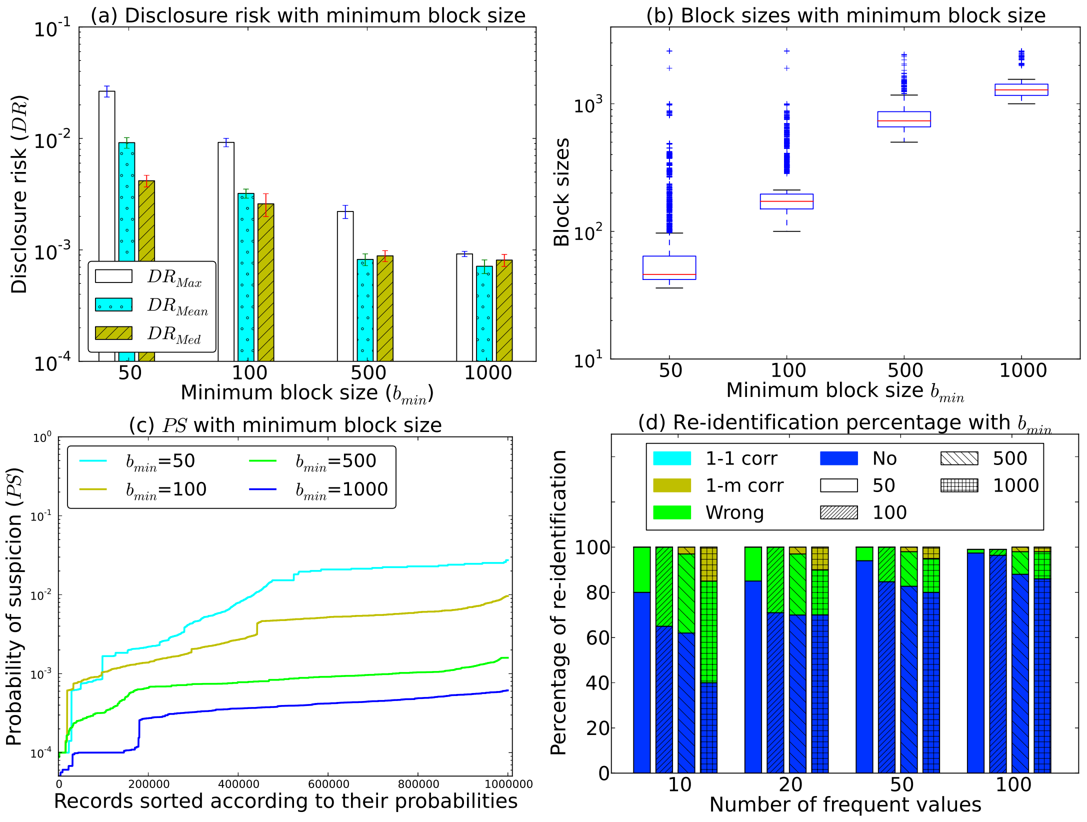


Figure 7.7: (a) disclosure risk values, (b) block sizes, (c) probability of suspicion (PS), and (d) the cryptanalysis attack results performed upon the generated blocks with minimum block size (b_{min}) for a 1,000K database. In here K represents 1,000 records. Note that plots have different y-axis scales.

As shown in Figure 7.7 (c), we computed average probability of suspicion (PS) values for a database with 1 million records for different b_{min}. Our distributed blocking approach has a maximum PS value less than ($1/b_{min}$) for each b_{min} value, which indicates a record in a block can be matched to more than b_{min} values in a global database $\mathbb{G}$ (under the worst case assumption of $\mathbb{G}$ being equal to the blocked database). As shown in this figure, our distributed blocking scheme provides strong privacy because most of the generated blocks are within the acceptable size limit.

Similar to Section 5.5, we conducted a cryptanalysis attack for each generated block of a database with 1 million records. We used the same parameter settings described in Section 5.5 to conduct this attack. As illustrated in Figure 7.7 (d), for smaller block sizes (b_{min} = 50 and 100) most of the re-identification guesses resulted in no guesses. We also noted that when b_{min} increases the number of 1-to-m (many) correct guesses also increases. However, it is important to note that 1-to-1 correct re-identifications are not possible for all b_{min} values. This indicates that an attacker could not re-identify an attribute value in a given block because it does not contain enough frequency information to identify q-grams that are encoded in BFs.

7.6 Summary

In this chapter we have proposed a scalable and efficient distributed blocking approach which allows each database owner to perform their blocking independently. This distributed blocking approach can be utilised in a linkage scenario where a linkage unit (LU) is available to facilitate the linkage. We used the LU to efficiently identify the blocks that need to be compared using a locality sensitive hashing technique. We evaluated our approach with large databases which indicates this approach is scalable with both the size and the number of databases. Next we describe an efficient way to identify the candidate block tuples that need to be compared across subgroups of database owners for the linkage situations where matches across subsets of records have to be found.

Chapter 8

Scalable Subgroup Blocking for Multidatabase Privacy-Preserving Record Linkage

As detailed in Chapter 1, the identification of similar blocks for different subgroups of databases is a primary challenge in any multidatabase privacy-preserving record linkage (MD-PPRL) application. In this chapter we propose a scalable subgroup blocking technique that can be used in both classical and privacy-preserving multi-database linkage applications. In Section 8.1 we discuss the importance and different challenges that occur when subgroups are to be identified. In Section 8.2 we will outline the main steps of our proposed approach, followed by a detailed description of each step in Section 8.3. In Section 8.4, we then provide a detailed analysis of our proposed approach in terms of complexity, blocking quality, and privacy. In Section 8.5 we describe the empirical study we conducted using large datasets and finally we summarise our findings in Section 8.6.

8.1 Introduction

In the previous Chapters 5, 6, and 7 we proposed several blocking techniques that can be used in a MD-PPRL context. However, these blocking techniques are only capable of generating candidate block tuples (CBTs) across all the databases that are to be linked. Neither of these techniques is capable of efficiently identifying CBTs across subgroups of databases. In real-world applications it would not be possible to identify sets of records that match in subgroups of databases if the linkage is performed across all databases. The sets of matching records across subgroups of databases can be valuable in conducting analytical studies and decision making about sub-populations or groups that exist within large entity populations [22].

For example, as we have described in Section 1.1 (on page 4), data analytic applications upon census databases are required to identify the sets of matching records across subgroups of census databases as these subset matching records are valuable for conducting analytical studies and decision making about sub-populations or

Table 8.1: Notation and terminology used in this chapter

$\mathbf{B}$	Set of blocks of a database owner
$\mathbf{BDP}$	List of block description pairs
$\mathbf{C}$	Set of cliques
$\mathbf{CBT}$	List of candidate block tuples
$\mathbf{CG}$	Set of candidate groups
$\mathbf{D}$	A database
$\mathbf{G}$	Candidate blocking graph
B	A block
B_i, B_j	Block identifiers
B_{rep}	A block representative
BDP	Block description pairs
C	A clique in graph $\mathbf{G}$
CBT	A candidate block tuple
E	List of edges
F	List of fixed databases
SG	A subgroup
V	List of vertices
d	Number of databases
e	An edge in a graph
g_α, g_β	Minimum and maximum subgroup size
n_r	Number of records in a database
sg	subgroup size
$sim()$	A similarity function
v_i, v_j	A vertex in a graph
w	Weight of an edge
w_t	Weight threshold
BKV	Blocking key value
DO	Database owner
LSH	Locality sensitive hashing
LU	Linkage unit
MD-PPRL	Multidatabase privacy-preserving record linkage

groups that exist within a larger population [41, 131, 188]. Another real-world example application would be a health surveillance system that continuously integrates and links data from hospitals and pharmacies to study the geographical and temporal effects of diseases and adverse drug reactions in certain patient groups [22, 24, 35]. Such analyses require subgroup linkage of such large databases collected over several years since the linkage across all databases would not be sufficient to identify subset matching records.

In order to identify the subsets of records that match in multiple databases, it must be possible to link records from subgroups of these databases. However, the usage of an existing blocking technique to identify similar blocks across subgroups in MD-PPRL is not computationally feasible for two reasons. (1) Existing blocking techniques for MD-PPRL are only capable of generating candidate blocks across all the databases, or for a subgroup of databases of a specific size [173, 183, 184]. (2)

The application of existing blocking techniques multiple times for linking subgroups of different sizes is computationally infeasible due to the large number of potential subgroup combinations to be considered in a MD-PPRL.

For example, with 5 databases, candidate block tuples (tuples of blocks of different databases that need to be compared, as defined in Section 4.2 on page 65) would be required for different subgroup combinations that consist of 2, 3, and 4 databases. More generally, assuming d databases are to be linked, the total number of subgroup combinations required is equal to $\sum_{|SG|=2}^{d} \binom{d}{|SG|}$, where $|SG|$ is the number of databases in a given subgroup SG. This makes MD-PPRL applications currently not scalable with regard to an increasing number of databases to be linked. Therefore, blocking techniques to be used in a MD-PPRL context should be capable of identifying similar records in subgroups of different sizes efficiently.

To address this problem, we propose a subgroup blocking approach for MD-PPRL which identifies blocks of records within a specific subgroup size range that need to be compared. As explained in Section 4.2, we utilise our proposed subgroup blocking approach in the second layer of our MD-PPRL blocking framework (Figure 4.1 on page 64). Our subgroup blocking approach accepts sets of blocks of the databases that are to be linked as input, and it generates a list of candidate block tuples for each selected subgroup that are to be compared in more detail using a private comparison and classification technique in the PPRL pipeline [35]. We next describe our approach in more detail. Table 8.1 summarises the notation we use in this chapter.

8.2 Overview of our approach

We assume d database owners (DOs) are participating in our subgroup blocking approach each with a database $\mathbf{D}$ to be linked, with $d \geq 2$. We use the notation $\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C$, $\mathbf{D}_D$, and so on for different databases. As described above the main aim of our approach is to generate candidate block tuples (CBTs) for different subgroups of different sizes. As defined in Chapter 4 on page 65, each candidate block tuple $CBT \in \mathbf{CBT}$ contains identifiers of blocks from between 2 and d databases, and a maximum of one block identifier per database. An overview of our subgroup blocking approach is shown in Figure 8.1.

Our proposed blocking approach follows the second linkage model described in Section 1.3 (on page 11) where we employ a *linkage unit* (LU) to facilitate the generation of CBTs for comparison. Similar to our distributed blocking scheme described in Chapter 7, to generate the CBTs across subgroups of databases, first each database needs to be blocked. We assume the blocking technique to be a *black box*, where any blocking technique can be used to generate the set of blocks for each database [35, 165]. In practice, blocking of all these databases can either be performed by the LU or locally by the database owners (DOs) on their own databases using our proposed blocking techniques described in Chapters 5, 6, and 7. Once the blocks are been generated, each DO sends a set of representatives of its blocks to the LU. We provide more details of the block representative generation in Section 8.3.

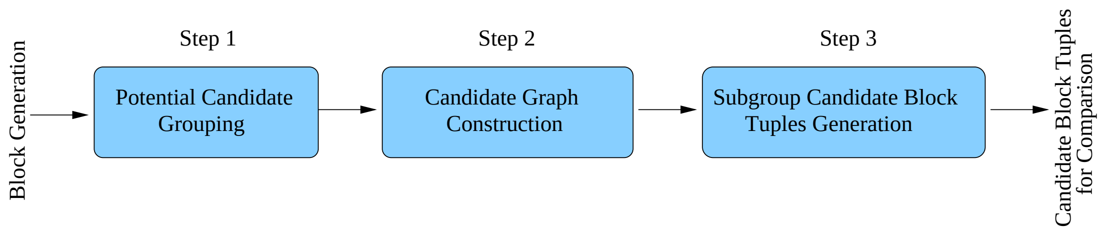

Block Generation
Step 1
Potential Candidate Grouping
Step 2
Candidate Graph Construction
Step 3
Subgroup Candidate Block Tuples Generation
Candidate Block Tuples for Comparison

Figure 8.1: Overview of our subgroup blocking with its three main steps. As detailed in Section 8.3, first the sets of blocks generated for each database are grouped into a set of candidate groups in step 1. In step 2, a graph is constructed by adding these blocks as vertices, where these vertices are connected using edges if their corresponding blocks are grouped in the same candidate group. In step 3, the candidate block tuples are identified for different subgroup combinations based on this graph.

To generate CBTs for subgroups of different databases, we introduce two user defined parameters, g_α and g_β, which limit the subgroup size range. g_α and g_β specify the minimum and maximum number of databases that can be included in subgroup combinations. As shown in Figure 8.1, our approach contains three steps:

1. *Potential Candidate Grouping*: To identify the blocks of different databases that are likely to be similar the LU uses a grouping technique upon the blocks of each database to generate a set of candidate groups ($\mathbf{CG}$). Based on the received block representatives, the LU assigns these representatives into candidate groups by using an appropriate blocking or grouping technique. We provide more details on this candidate grouping in Section 8.3. A candidate group $CG \in \mathbf{CG}$ helps to identify the potential candidates that need to be considered for comparison among the sets of blocks from different databases. This grouping also reduces the overall number of block comparisons since only the blocks that are grouped into a CG will be compared next in the comparison and classification step of the PPRL process. Reducing the number of block comparisons therefore removes record comparisons that are unlikely to be matched.

2. *Candidate Graph Construction*: A candidate graph $\mathbf{G}$ is constructed based on the generated candidate groups $\mathbf{CG}$. The LU iterates over each candidate group CG in $\mathbf{CG}$ to create vertices and edges in $\mathbf{G}$. The blocks from each $CG \in \mathbf{CG}$ become vertices, and a pair of vertices is connected by an edge if both vertices appear in the same CG. Then the LU updates the candidate graph into a weighted $\mathbf{G}$ by computing a weight for each edge. The weights are calculated based on the similarity between the block representatives, as we discuss in detail in the following sub-section.

3. *Subgroup Candidate Block Tuple Generation*: In this step the LU generates CBTs for each subgroup combination using $\mathbf{G}$. As we explain below, a *weight threshold* is used to prune low weighted edges (pairs of blocks) in $\mathbf{G}$. The pruning of

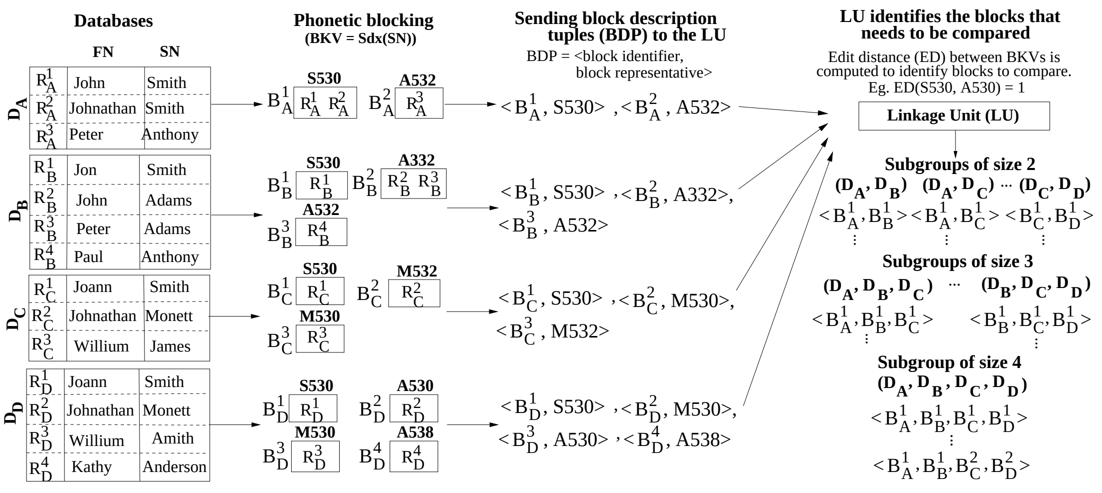


Figure 8.2: An illustrative example of distributed blocking of four databases using Soundex based blocking. The values in the *surname* attribute (SN) are used as blocking key values (BKVs). Each database, $\mathbf{D}_A$ to $\mathbf{D}_D$, is blocked independently and the linkage unit (LU) identifies the blocks that need to be compared between subgroups of different databases based on the edit distance between Soundex encodings of the BKVs (with maximum edit distance of 1).

low weighted edges ensures any pair of blocks that has a low similarity is not considered in the *CBT* generation for any subgroup combination that includes the corresponding databases of the blocks in these pairs.

To identify the *CBT*s that need to be compared across subgroups using the constructed graph **G**, we introduce a novel constraint-based algorithm that uses a depth-first search over **G**. Based on the parameters g_α and g_α our subgroup blocking approach is capable of generating *CBT*s for all possible subgroup combinations across databases that need to be linked. Optionally, we also allow the user to define the set F of databases that must be included in every subgroup to be generated. We next describe our subgroup blocking approach in more detail.

8.3 Scalable Subgroup Blocking Approach

Without loss of generality let us assume each database is blocked independently by the owner (DO) of the database, and the same technique is used to perform blocking by each DO. Such blocking provides flexibility and efficiency over the block generation process and also ensures the same grouping strategy will be used when blocking the different databases, as we described in Chapter 7.

However, in practice any blocking technique such as those described in Chapters 5 to 7 can be used to generate blocks. For illustrative purposes, a simple block generation process using phonetic blocking [35] is shown in Figure 8.2. We use

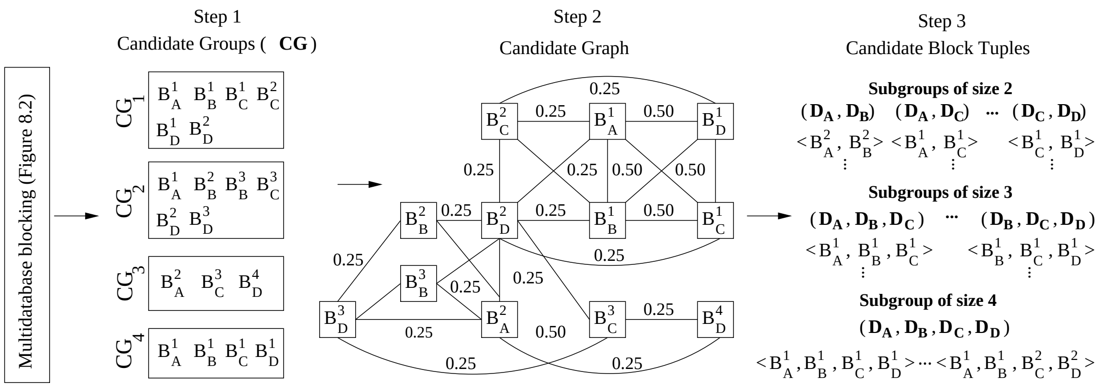


Figure 8.3: An illustrative example of our subgroup blocking approach with its three main steps. In this example, the sets of blocks generated in Figure 8.2 are assigned into the set of candidate groups (**CG**) in step 1. As we explain in Algorithm 8.1, a candidate graph **G** is constructed based on **CG**. In this example, each edge is assigned with a normalised weight that corresponds to the number of groups in **CG** that contain a given pair of blocks. In step 3, the candidate block tuples are identified for different subgroup combinations. In this example, the minimum and maximum subgroup sizes (g_α, g_β) are set to $(2, 4)$.

the notation $\mathbf{B}_A$, $\mathbf{B}_B$,$\mathbf{B}_C$, and $\mathbf{B}_D$ to represent the sets of blocks generated for the databases $\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C$, and $\mathbf{D}_D$, respectively.

Similar to Chapter 7, each DO generates a *block description pair* for each of its blocks. We consider the generation of block description pairs as a *black box*. Each block description pair $BDP = (B_i, B^i_{rep})$ consists of a block identifier (B_i) and a corresponding block representative (B^i_{rep}). We use the notation B_A, B_B, B_C, and B_D to represent B_is of blocks of $\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C$, and $\mathbf{D}_D$, respectively.

A B_{rep} can be generated in different forms, such as a Min-Hash signature [101] (as we have described in Section 7.3 on page 136), a Bloom filter [190], or a phonetic encoding [35], as long as the same technique is used by all DOs to generate B_{rep}s that allow the calculation of similarities between the pairs of blocks that need to be compared across databases. As shown in Figure 8.2, the phonetic encoding of BKVs of the surname attribute is used as B_{rep}s for the generated blocks.

Next, each DO sends its set of BDPs to the LU. These generated BDPs are added to a set **BDP** to identify the blocks that need to be compared, across different subgroups of databases using appropriate comparison and classification techniques [35, 210, 216]. As illustrated in Figure 8.2, based on the edit distance [35] between the B_{rep}s of the blocks of different databases, the LU generates a set of CBTs for comparison for each subgroup of different sizes.

An illustrative example of subgroup blocking for four databases $\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C$, and $\mathbf{D}_D$ with (g_α, g_β) set to (2,4) is shown in Figure 8.3. As can be seen, the CBTs are generated for different group combinations across these four databases for subgroup sizes 2, 3, and 4. Based on the parameters (g_α, g_β), our subgroup blocking is capable of generating CBTs for the following three different scenarios with d databases:

1. $g_\alpha = g_\beta = d$: This setting gives the linkage between all databases only, i.e. only matching sets of blocks across all databases are generated. Following the example in Figure 8.3, if (g_α, g_β) = (4, 4) then *CBTs* are generated only for subgroup combination ($\mathbf{D}_A$, $\mathbf{D}_B$, $\mathbf{D}_C$, $\mathbf{D}_D$).

2. $2 \leq g_\alpha \leq g_\beta < d$: This setting gives all possible linkages for subgroups with blocks from at least g_α to at most g_β databases. As illustrated in Figure 8.3, if (g_α, g_β) is set to (2, 4) then all subgroup combinations of size 2 and 3 across databases $\mathbf{D}_A$, $\mathbf{D}_B$, $\mathbf{D}_C$, and $\mathbf{D}_D$ are considered in CBT generation.

3. $F = \{\mathbf{D}_i, \mathbf{D}_j, \cdots, \mathbf{D}_x\}$, $2 \leq g_\alpha \leq g_\beta < d$: This setting generates candidate tuples for subgroups with size at least g_α to maximum size of g_β out of d databases, where databases $\mathbf{D}_i, \mathbf{D}_j, \cdots, \mathbf{D}_x$ are fixed and $|F| \leq g_\alpha$. In this scenario, databases $\mathbf{D}_i, \mathbf{D}_j, \cdots, \mathbf{D}_x$ must appear in every subgroup combination. As illustrated in Figure 8.3, for example, if $F = \{\mathbf{D}_A\}$ and $(g_\alpha, g_\beta) = (2, 3)$ then the subgroup combinations that will be considered in our approach are, for subgroups of size 2: $(\mathbf{D}_A, \mathbf{D}_B)$, $(\mathbf{D}_A, \mathbf{D}_C)$, and $(\mathbf{D}_A, \mathbf{D}_D)$, and for subgroups of size 3: $(\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C)$, $(\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_D)$, and $(\mathbf{D}_A, \mathbf{D}_C, \mathbf{D}_D)$.

Each of these scenarios represents a real-world situation that can occur in a MD-PPRL context. In Section 8.5, we will experimentally evaluate our subgroup blocking approach for each of these scenarios. We next describe each step in our proposed subgroup blocking approach in more detail.

1. **Potential Candidate Grouping**

   As shown in Figure 8.1, in the first step the LU identifies the potential candidates among the sets of blocks of each database by grouping the corresponding BDPs into a set of candidate groups (**CG**). We consider the generation of **CG** to be a *black box* as the grouping technique depends on the B_{rep}s generated for all databases.

   For example, a Jaccard based locality sensitive hashing (LSH) [101, 116] technique (as discussed in Chapter 7) can be used with B_{rep}s based on Min-Hash signatures, where B_{rep}s that hash to the same bucket become candidates and each bucket is considered as a CG and is added to **CG**. As an alternative (as illustrated in Figure 8.2) the blocks with the same or similar phonetic encodings can be added to the same CG.

   Each group $CG \in \mathbf{CG}$ contains similar BDPs from different databases. The BDPs in a CG can be used to generate different CBT combinations, as potential candidates, for subgroup comparisons. For example in Figure 8.3, B^1_A, B^1_B, B^1_C, and B^1_D in CG_1 generate (B^1_A,B^1_B), (B^1_A,B^1_C), (B^1_A,B^1_D), (B^1_B,B^1_C), (B^1_B,B^1_D), (B^1_C,B^1_D), $\langle B^1_A, B^1_B, B^1_C \rangle$, $\langle B^1_A, B^1_C, B^1_D \rangle$, $\langle B^1_A, B^1_B, B^1_D \rangle$, and $\langle B^1_B, B^1_C, B^1_D \rangle$ as potential candidates for comparison for subgroups of sizes 2 and 3, respectively, because B^1_A, B^1_B, B^1_C, and B^1_D are likely to be similar.

Algorithm 8.1: Candidate graph construction

```
Input  : CG - Set of candidate groups
Output: G   - Candidate graph
1  $V \leftarrow \emptyset, E \leftarrow \emptyset$                    // Initialise an empty graph
2  foreach $CG \in \mathbf{CG}$ do                                   // Iterate over every $CG$
3      foreach $(B_i, B^i_{rep}) \in CG$ do                          // Add each $BDP$ to graph
4          $v \leftarrow B_i$                                       // Create a new vertex
5          $V.add(v)$                         // Add the newly created vertex to graph
6          foreach $(B_j, B^j_{rep}) \in CG$ do        // Add each candidate block to graph
7              if $B_i \neq B_j \wedge (B_i, B_j) \neq E$ then   // Check the edge is already created
8                  $E.add((B_i, B_j))$                              // Create a new edge
9                  $(B_i, B_j).w \leftarrow calcWeight((B_i, B^i_{rep}), (B_j, B^j_{rep}))$   // Compute weight $w$
10             end
11         end
12     end
13 end
14 return $\mathbf{G} = (V, E)$
```

If a group of blocks appears in multiple CGs it is more likely that these blocks are more similar. In Figure 8.3 (step 1) for example, the pair (B^1_A,B^1_C) is more likely to be similar compared to the pair (B^1_A,B^2_C), because (B^1_A,B^1_C) occurs in two CGs while (B^1_A, B^2_C) only occurs in one CG in **CG**.

2. **Step 2 - Candidate Graph Construction**

   As shown in Figure 8.3, the second step of our approach is to construct the candidate graph $\mathbf{G} = (V, E)$ from **CG**. We use a graph data structure to represent similarity relationships between the blocks from different databases. We consider each block as a vertex in the graph where the set of edges between those vertices represents the comparisons between blocks. In this approach we use a d-partite graph [1] since the vertices in **G** generally represent different blocks of d databases. We formally define the candidate graph **G** as follows.

   Definition 8.1. Candidate Graph **G**
   Given a set **CG** of candidate groups, the undirected candidate graph derived from **CG** is a d-partite graph $\mathbf{G} = (V, E)$, where V is a set of vertices and E is a set of undirected edges that each is an unordered pair of elements of V. V represents all unique blocks in **CG**, i.e. $V = \{\cup\, B \in CG : \forall\, CG \in \mathbf{CG}\}$, while E contains all undirected edges between pairs of vetices such that $E = \{(B_i, B_j) : (B_i \in CG) \wedge (B_j \in CG) \wedge (i \neq j), \forall\, CG \in \mathbf{CG}\}$.

   The construction of **G** is outlined in Algorithm 8.1. First the LU iterates over each candidate group CG in the set **CG** generated in step 1 of our approach (line 2). For each block appearing in a $CG \in \mathbf{CG}$ its corresponding block identifier (B_i) is added into the graph **G** as a new vertex $v \in V$ (lines 3 and 4).

An edge $e \in E$ is created between two vertices, v_i and v_j, if their corresponding blocks B_i and B_j appear in a CG, such that edges are created only between blocks from different databases (lines 6 to 8).

As shown in Figure 8.3 (step 2), a normalised weight w is calculated, $0 \leq w \leq 1$, for each edge (B_i, B_j) using the function *calcWeight()* (line 9). The weight w of edge (B_i, B_j) can be computed in different ways based on the generated BDPs, such as the similarity between the corresponding block representatives B^i_{rep} and B^j_{rep}, or the cardinality of (B_i, B_j) which is $w = \{|\{CG : \forall_{CG \in \mathbf{CG}}\ (B_i, B_j) \in CG\}|\}$ / $|\mathbf{CG}|$, where $|\cdot|$ represents the cardinality of a given set.

We assume B_{rep}s (that depend upon the actual blocking technique used) can be compared using an appropriate similarity function $sim(B^i_{rep}, B^j_{rep})$ to measure their similarity. For example, the Jaccard coefficient [216], Dice coefficient [216], and edit distance [35] can be used with Min-Hash signatures [25], Bloom filters [190], and phonetic encodings [35], respectively. These weights are used in the next step of our approach to generate candidate block tuples.

3. **Subgroup Candidate Generation**

   In third step of our subgroup blocking approach, the LU generates candidate block tuples (CBT) for each subgroup combination required, as illustrated in Figure 8.1. For a given subgroup combination of size g_α a CBT contains a maximum of one block per database, and blocks from at least g_α databases. We consider each CBT to be a *clique* $C \in \mathbf{G}$, where each $C \subseteq V$, such that all pairs of vertices in C are connected by an edge, i.e., $\forall v_i, v_j \in C : \ (v_i, v_j) \in E$.

   To control the number of CBTs generated for each subgroup, we use a constraint on the weight w of each edge $e \in E$ which we name the *weight threshold*, w_t. This weight constraint specifies the minimum weight that e must have to be considered in the clique generation. The weight constraint helps to control the density of $\mathbf{G}$ by efficiently pruning edges with weights lower than w_t. This pruning of edges of $\mathbf{G}$ ensures block pairs with weights below w_t are not being considered. In practice, different w_ts can be specified for different subgroup database combinations depending on the DOs' requirements.

   Depth-first based Candidate Generation

   In order to identify the set of CBTs for a given subgroup size g_α, we propose a depth-first algorithm where the set of CBTs of a given subgroup combination is generated using a depth first traversal through the graph $\mathbf{G}$ [1]. As outlined in Algorithm 8.2, this depth-first based candidate generation approach uses an iterative deepening depth-first search algorithm [182] which generates CBTs from size g_α to g_β by incrementally expanding the size of subgroups.

   The depth-first candidate generation approach is based on a multi-branch recursion that allows the graph $\mathbf{G}$ to be searched in a depth-first manner, where the algorithm progressively searches for similar blocks for a given subgroup

Algorithm 8.2: Depth-first based candidate generation

```
Input  : G      - Undirected candidate graph
         $w_t$     - Weight threshold
         $g_\alpha$     - Minimum subgroup size
         $g_\beta$     - Maximum subgroup size
         $d$      - Number of databases
         $F$      - Set of fixed databases
         BDP - Set of block description pairs
Output: CBT - Inverted index of subgroup candidate block tuples

CBT ← {}, SG ← {}                                    // Initialise variables
$K.add(genSubgroupComb(g_\alpha, g_\beta, d, F))$               // Generate subgroup combinations
foreach $sg \in$ SG.$keys()$ do                                // Process each subgroup size
    foreach $SG \in$ SG[$sg$] do            // Process each subgroup combination of size $sg$
        CBT[$SG$].$add(genCandidates($G, $SG, w_t$, BDP))           // Generate block tuples
    end
end
return CBT

Function genCandidates(G, $SG, w_t$, BDP):
    C ← []                                  // Initialise a list to hold block tuples
    if $SG = 2$ then                                 // Check subgroups size equals 2
        C ←$getEdges($G, $SG, w_t)$                       // Get edges with weight $\geq w_t$
        return C
    end
    else
        D, $BL \leftarrow getDBWithMinBlocks($BDP, $SG)$           // Get D with minimum |BDP|
        foreach $(B_i, B^i_{rep}) \in BL$ do                       // Iterate over every $BDP$
            $L \leftarrow getNeighbours($G, $\{subgroup -$ D$\}, w_t, B_i)$   // Get neighbour vertices
            $C \leftarrow genCandidates($G, $\{subgroup -$ D$\}, w_t, L)$     // Generate block tuples
            C.$add(updateCandidates(C, B_i))$          // Concatenate $B_i$ with generated tuples
        end
    end
    return C
end
```

combination of databases based on the similarities between the corresponding B_{rep}s until the required subgroup size is reached.

Our proposed depth-first candidate generation algorithm starts by generating all the required subgroup combinations to be considered in the candidate generation for all databases using the function *genSubgroupComb()* (line 2 in Algorithm 8.2). This function accepts the minimum and maximum subgroup size g_α and g_β, the number of databases, and the set of fixed databases (F) as inputs and computes the required subgroup combinations for each subgroup size from g_α to g_β. These generated combinations are added to an inverted index **SG** (initialised in line 1) where the subgroup sizes are used as keys and possible combinations are added as values.

For example, with $(g_\alpha, g_\beta) = (2, 3)$ of a linkage between databases $\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C$, and $\mathbf{D}_D$, $\mathbf{SG}[2]$ contains the list of combinations $(\mathbf{D}_A, \mathbf{D}_B)$, $(\mathbf{D}_A, \mathbf{D}_C)$, $(\mathbf{D}_A, \mathbf{D}_D)$, $(\mathbf{D}_B, \mathbf{D}_C)$ and $(\mathbf{D}_B, \mathbf{D}_D)$, and $(\mathbf{D}_C, \mathbf{D}_D)$ while $\mathbf{SG}[3]$ contains the group combinations $(\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C)$, $(\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_D)$, $(\mathbf{D}_A, \mathbf{D}_C, \mathbf{D}_D)$, and $(\mathbf{D}_B, \mathbf{D}_C, \mathbf{D}_D)$.

For each subgroup combination in **SG** the recursive function *genCandidates()* is called to generate the set of *CBT*s (in lines 3 to 7). The function *getEdges()* identifies the trivial cliques for each subgroup SG of size $sg = 2$ which are the set of edges $e \in E$ of **G** for the pair of databases that satisfy the weight threshold w_t (lines 11 to 14 in Algorithm 8.2). In line 5, these computed edges are added to the inverted index **CBT** using each corresponding subgroup combination as a key.

For subgroup sizes greater than 2, the function *genCandidates()* selects the database **D** with the minimum number of blocks BL using a function *getDBWithMinBlocks()*. The selection of **D** with the minimum number of blocks is important in Algorithm 8.2 because it decides the maximum number of recursive *genCandidates()* executions required in the CBT generation when processing each database in a given subgroup (line 16). Assuming a subgroup combination ($\mathbf{D}_A$, $\mathbf{D}_B$, $\mathbf{D}_C$, $\mathbf{D}_D$) for example, $\mathbf{D}_A$ is selected first because it only has two blocks (as shown in Figure 8.2) to be considered in the CBT generation that require only two *genCandidates()* calls. Next, for each (B_i, B^i_{rep}) in BL, the function *getNeighbours()* computes the neighbouring vertices for the remaining set of databases that connect with B_i (line 18 in Algorithm 8.2). Those pairs of vertices that have an edge weight greater than or equal to w_t are added to the list L.

For each BDP in BL the function *genCandidates()* is called recursively with the list of neighbouring nodes, w_t, and the set of remaining databases as inputs (line 19). Each of these recursive calls returns a list of block tuples C, where each tuple in C is updated with the current processed BDP using the function *updateCandidates()* (line 20). As shown in Figure 8.3, for block B^1_A of database $\mathbf{D}_A$ for example, if the recursive call returns a list of block pairs $[(B^1_B, B^1_C), (B^1_B, B^2_C), \cdots]$ between $\mathbf{D}_B$ and $\mathbf{D}_C$ then each block pair is concatenated with B^1_A to generate tuples $\langle B^1_A, B^1_B, B^1_C \rangle$, $\langle B^1_A, B^1_B, B^2_C \rangle$, and so on. This allows Algorithm 8.2 to progressively generate the cliques until they reach the required size g_α. These generated CBTs are finally added to an inverted index **CBT** using subgroups as their keys (line 5 in Algorithm 8.2).

8.4 Conceptual Analysis of Subgroup Blocking

We analyse our subgroup blocking approach in terms of complexity and blocking quality. We also analyse the privacy of our approach assuming it is employed in a privacy-preserving context. We assume d databases are to be linked in our approach. Let us assume each database is blocked using the same blocking technique and a list of blocks **B** is being formed for each database.

8.4.1 Complexity

In step 1 of our subgroup blocking approach (as shown in Figure 8.1), the complexity of the approach depends on the technique used to generate the set of candidate groups ($\mathbf{CG}$). We assume each candidate group $CG \in \mathbf{CG}$ contains d BDPs from different databases. In step 2, the $\mathbf{CG}$ is used to construct the candidate graph $\mathbf{G}$. This requires an iteration through each $CG \in \mathbf{CG}$ to add a vertex v to $\mathbf{G}$, and to create edges between vertices if they share the same CG. A CG with d BDPs generates $\frac{d(d-1)}{2}$ edges in $\mathbf{G}$. Hence, the construction of $\mathbf{G}$ has a complexity of $O(\mathbf{CG} \cdot d^2)$.

Based on the values of g_α and g_β, in step 3 the proposed depth-first based candidate generation technique (Algorithm 8.2) requires to generate CBTs for $\sum_{g_\alpha=2}^{g_\beta} \binom{d}{g_\alpha}$ subgroup combinations. For each combination, Algorithm 8.2 uses a multi-branch recursion to generate CBTs. In the function *genCandidates()*, at each recursion a database $\mathbf{D}$ out of g_α databases with the minimum number of vertices is selected (line 16 of Algorithm 8.2). Without loss of generality, let us assume the number of BDPs selected for $\mathbf{D}$ is m. m defines the maximum recursion branch factor of $\mathbf{D}$. The total number of vertex traversals in $\mathbf{G}$ for a given subgroup combination of size g_α is $g_\alpha \cdot m + (g_\alpha - 1) \cdot m^2 + \cdots + 2 \cdot m^{g_\alpha - 1} + m^{g}_{\alpha}$ which is $\sum_{i=0}^{g_\alpha}(g_\alpha + 1 - i) \cdot m^i$. Hence, for $\sum_{g_\alpha=2}^{g_\beta} \binom{d}{g_\alpha}$ subgroup combinations the proposed depth-first candidate generation algorithm has a complexity of $O(\sum_{g_\alpha=2}^{g_\beta} \sum_{i=0}^{g_\alpha} \binom{d}{g_\alpha}(g_\alpha + 1 - i) \cdot m^i)$. However, this algorithm would require a worst-case complexity of $O(\sum_{g_\alpha=2}^{g_\beta} \binom{d}{g_\alpha} \cdot |\mathbf{B}|^{g}_{\alpha})$ if $\mathbf{G}$ is a complete graph with $\forall e \in E: \ e.w \geq w_t$.

In our approach, if databases from several organisations are to be linked, then communication only occurs once when the CBTs are generated and sent to the LU. For a given subgroup SG of size g_α our approach will send a message to each DO in SG with the list $\mathbf{C}$ of generated CBTs. Therefore, the entire subgroup blocking approach has a communication complexity of $O(\sum_{g_\alpha=2}^{g_\beta} \binom{d}{g_\alpha} \cdot g_\alpha)$ messages each of size $|\mathbf{C}|$, where $\mathbf{C}$ contains tuples of blocks of size $|SG|$.

8.4.2 Blocking Quality

We analyse the quality of our subgroup blocking in terms of efficiency and effectiveness. The overall quality depends upon the blocking process and the generated block description pairs (BDPs) which are considered as *black boxes* in our approach. In step 1 of our approach the effectiveness of the candidate grouping depends on the B_{rep}s and the accuracy of the grouping technique used on those B_{rep}s.

In step 2, the density of graph $\mathbf{G}$ depends on the number of different blocks in each candidate group $CG \in \mathbf{CG}$ from different databases to be linked. A large number of blocks in a CG will increase the effectiveness of the CBT generation process as more edges are created in $\mathbf{G}$. However, this would increase the runtime of candidate graph construction as more blocks are added into $\mathbf{G}$ as vertices and new edges are created between these vertices when the LU iterates through each $CG \in \mathbf{CG}$.

In step 3 of our approach, the weight threshold w_t provides a trade-off between quality and efficiency of the candidate block tuple generation process. The threshold

w_t is used to prune the edges (block pairs) with weights lower than w_t. A lower w_t will generate more cliques (CBTs) as more edges are considered in the candidate block tuple generation process for a given subgroup. This will increases the number of true matches [35] as more candidate block tuples are generated to be compared in the comparison and classification step, which improves the effectiveness of the overall linkage. However, a lower w_t will potentially also increase the overall runtime and space requirements of our approach since more block pairs are considered for a given subgroup combination.

8.4.3 Privacy

We next analyse the privacy of our subgroup blocking approach. We assume each party (DOs and the LU) participating in our approach follows the honest-but-curious (HBC) adversary model [212] as discussed in Section 2.4 (on page 27).

We assume each DO generates its blocks independently by using a private blocking technique as described in Chapter 7. Besides an initial agreement of parameter settings to be used, this technique does not require any further communication between the DOs that would reveal information about their blocks. We also assume each block contains at least $k = \frac{n_r}{|\mathbf{B}|}$ records, where n_r is the total number of records in a database $\mathbf{D}$ (assuming all databases are of size $n_r = |\mathbf{D}|$) and $|\mathbf{B}|$ is the number of blocks generated for $\mathbf{D}$, to ensure k-anonymous privacy [200]. Therefore, no DO can learn anything about any other DO's sensitive data during the blocking step.

In our approach each DO sends a list of block description pairs (BDPs) to the LU where each pair contains a block identifier and a block representative (B_{rep}). Apart from this information the LU does not learn anything regarding the encoding methods used for records and the parameter values used in the blocking techniques used by the DOs. As discussed in Section 8.2, a B_{rep} can be generated in different forms while the parameter setting used in the generation of B_{rep}s is also unknown to the LU.

If the LU is conducting the linkage of pairs of blocks sent to it by the DOs, it potentially can conduct frequency attacks on the masked records, the block sizes, as well as the number of matched records. However, as has been shown in [132], even sophisticated cryptanalysis attacks based on a constrained satisfaction solvers that can learn information about some individual records in a block are only successful for certain parameter settings of the encoding methods used.

Because each DO can perform a generalisation strategy (k-anonymisation) [200] on its blocks independently, and each database likely contains different frequency distributions of values, a frequency attack based on block sizes becomes more difficult for the LU to conduct compared to a frequency attack on a single set of blocks only. Additionally, encoding techniques that use record level rather than attribute level encodings [56, 216] can be employed to further secure the overall PPRL process and reduce the risk of information leakage to the LU.

Table 8.2: Parameter settings used in the experimental evaluation

Parameter	Value range
Dataset name	NC-SYN
Number of databases (d)	3, 5, 7, 10
Number of records in each database (n_R)	5,000 to 1,000,000
Blocking key attributes (A)	Given name, Surname, City (suburb), and Postcode
Corruption levels	0%, 20%, and 40%
Minimum subgroup size (g_α)	2
Maximum subgroup size (g_β)	3 to 10
Number of fixed databases ($\|F\|$)	1 to 4
Weight threshold (w_t)	0.2 to 0.8
Number of candidate groups ($\|\mathbf{CG}\|$)	100, 150, 200, 250

However, under the HBC model collusion is possible between the participating parties. Any collusion between a DO and the LU would increase the risk of frequency attacks on another DO's database. The colluding DO can reveal the parameter settings used in blocking and the B_{rep}s generation process to the LU. However, as we have shown in Section 7.5, since each DO performs its blocking independently and does not share any information about its set of blocks it is not possible to conduct a frequency analysis on a database of a non-colluding DO.

8.5 Experimental Evaluation

In this section, we present and discuss the results of the experimental evaluation study of our subgroup blocking approach conducted on the datasets described in Section 4.4 (on page 78). Table 8.2 summarises the parameter values used in our approach. We use the distributed blocking scheme proposed in Chapter 7 to generate the blocks for all databases. We use the same initial steps and parameter settings as in Table 7.2 (on page 149) to generate the blocks. We use Min-Hash signatures as block representatives (b_{rep}s) in block description pairs (BDPs). These b_{rep}s are then hashed into a set of buckets using locality sensitive hashing (LSH). We follow the same parameter settings and steps described in Section 7.3 (on page 136) to generate these b_{rep}s and to conduct LSH. Each bucket is added to the list **CG** as a candidate group. To measure the similarity between the b_{rep}s in step 2 of our approach we use the Jaccard coefficient [35].

8.5.1 Scalability

As shown in Figures 8.4 (a) and (b), we measure the average runtime for potential candidate grouping and candidate graph construction of our approach, respectively, with different number of databases (d). As can be seen in Figure 8.4 (a), the average runtime for candidate grouping increases linearly with d because more block description pairs (BDPs) of different databases need to be hashed into buckets in LSH. This figure also shows that the average runtime increases with the number of BDPs which

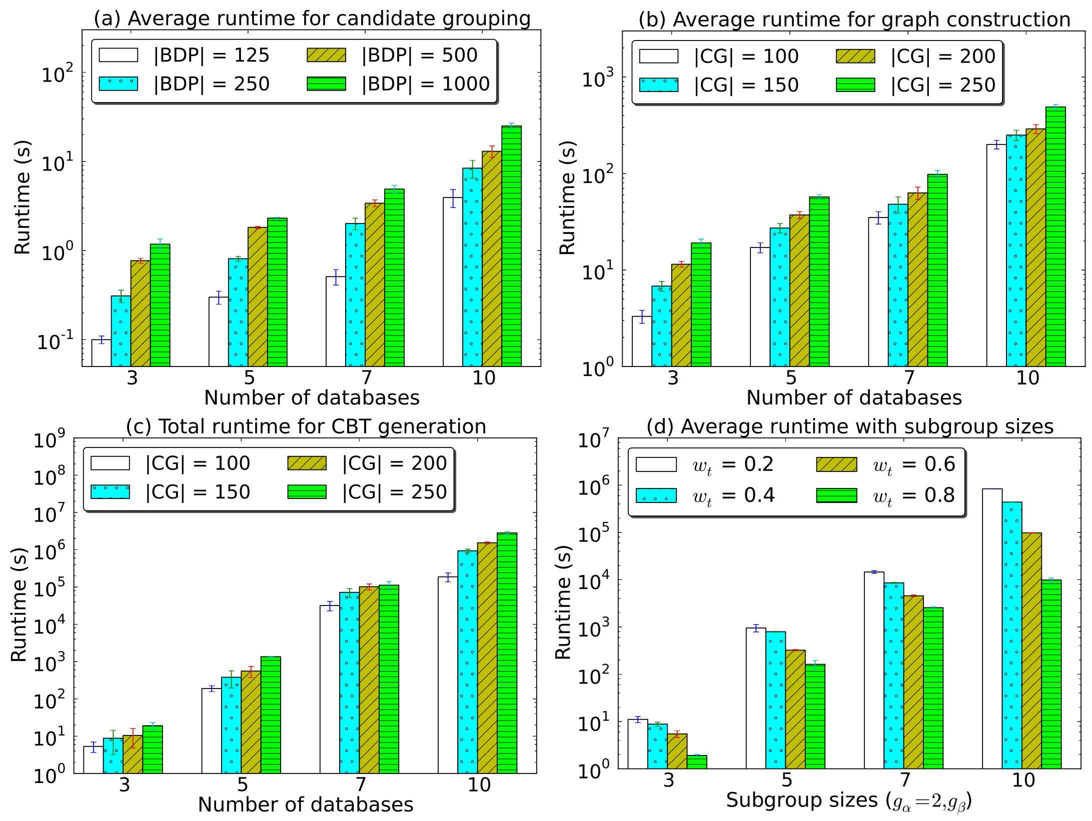


Figure 8.4: Average runtime for (a) candidate grouping for different number of block description pairs (BDPs) and (b) candidate graph construction for different number of candidate groups for different number of databases, d. (c) and (d) show the total runtime with different d and average runtime with different subgroup sizes for CBT generation, respectively. Note that plots have different y-axis scales.

depends on the block generation process by each databases owner (DO). However, it took only 26.7 seconds to hash 1,000 BDPs of 10 databases.

Figure 8.4 (b) shows the average runtime required for the candidate graph ($\mathbf{G}$) construction with different numbers of candidate groups ($|\mathbf{CG}|$) for different numbers of databases. As can be seen in this figure, the average runtime increases linearly with $|\mathbf{CG}|$ because the LU iterates through each $CG \in \mathbf{CG}$ in Algorithm 8.1. This figure also shows that the construction of $\mathbf{G}$ requires more time with an increasing number of databases which suggests that more BDPs are added to $\mathbf{G}$ as vertices and more block pairs are being generated across these databases, which increases the number of edges in graph $\mathbf{G}$.

Figures 8.4 (c) and (d) illustrate the total and average runtime required for the CBT generation with different numbers of databases and subgroup sizes, respectively. As shown in Figure 8.4 (c), total runtime increases exponentially as the number $\sum_{g_\alpha=2}^{d} \binom{d}{g_\alpha}$ of subgroup combinations grows exponentially with d (while the runtime grows linearly with d for a given subgroup size). We also noted that the runtime

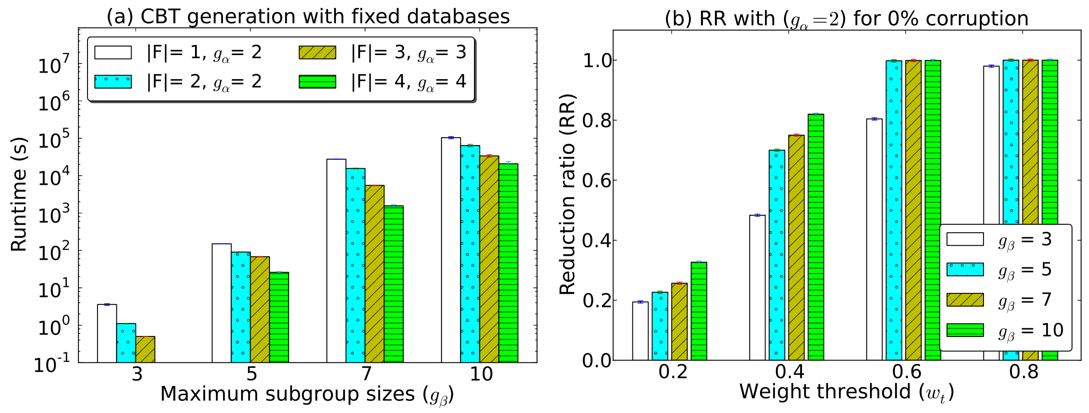


Figure 8.5: (a) Average runtime for CBT generation with different fixed number databases ($|F|$) and (b) reduction ratio with different weight thresholds (w_t) for different maximum subgroup sizes (g_β). Note that plots have different y-axis scales.

increases linearly with $|CG|$ because more BDPs are added into graph **G** and more edges are considered in generating CBTs in Algorithm 8.2.

As expected the average runtime decreases with an increase in the weight threshold (w_t) as shown in Figure 8.4 (d). As the weight constraint w_t increases, edges with lower similarity between their corresponding b_{rep}s are not considered in the CBT generation. However, the runtime increases linearly with the size of subgroups as more subgroup combinations are considered in CBT generation.

Figures 8.5 (a) and (b) show the average runtime required for generating CBTs for different subgroup sizes with a set of databases (F) fixed and reduction ratio (RR) with w_t for different maximum subgroup sizes (g_β). As expected, the average runtime decreases when more databases are included in F since the number of subgroup combinations considered in each subgroup size decreases with the size of F, as shown in Figure 8.5 (a). As shown in Figure 8.5 (b), RR increases with w_t which suggests that fewer CBTs are generated for each subgroup combination considered in Algorithm 8.2 for a given maximum subgroup size g_β.

8.5.2 Blocking Quality

Similar to previous chapters we measure the pairs completeness (PC) and F-measure (FM) to evaluate the effectiveness of our subgroup blocking approach. Figures 8.6 (a) and (b) show the PC with w_t and $|\mathbf{CG}|$ for different g_β and d, respectively. As shown in Figures 8.6 (a), a lower w_t value increases PC by generating more CBTs at each iteration in Algorithm 8.2, which increases the overall runtime of our approach (see Figure 8.4 (d)). However, as can be seen in Figure 8.5 (b), a lower w_t increases the overall number of candidate record tuple (CRT) comparisons which decreases RR of our subgroup blocking approach.

Figure 8.6 (b) shows average PC for different numbers of databases for different subgroup combinations with subgroup sizes (g_α, g_β) set to (2, 3), respectively. PC

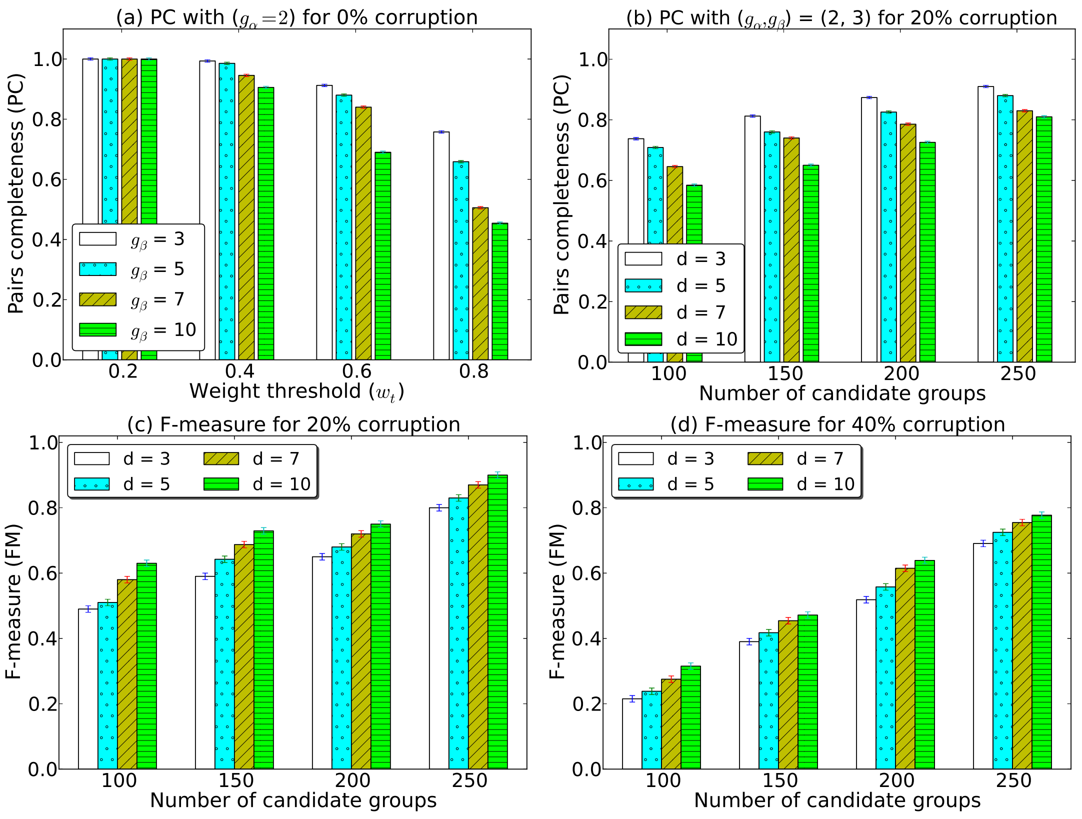


Figure 8.6: Pairs completeness (PC) with (a) weight threshold (w_t) for different maximum subgroup size (g_β) and (b) different number of databases (d) for different number of candidate groups ($|\mathbf{CG}|$). F-measure (FM) with different number of databases for different $|\mathbf{CG}|$ with (c) 20% and (d) 40% corrupted databases.

increases with $|\mathbf{CG}|$ because more block pairs (edges) are considered in the *CBT* generation process. However, we note that PC decreases with the number of databases. This is because some true matches are missed due to the low similarities between corresponding B_{rep}s. This suggests that by using more candidate groups in step 1 our approach can achieve high PC values even when the databases have low quality data.

Figures 8.6 (c) and (d) show the FM with different corruption levels for different number of candidate groups. FM increases with $|\mathbf{CG}|$ which suggests that *CBT* generation becomes more fine grained as the candidate graph $\mathbf{G}$ becomes more dense. However, an increment of $|\mathbf{CG}|$ could potentially increase the runtime of our approach as shown in Figure 8.4 (c). As can be seen in these figures, FM decreases with an increase in the corruption level which suggests that our approach is affected by the quality of the data.

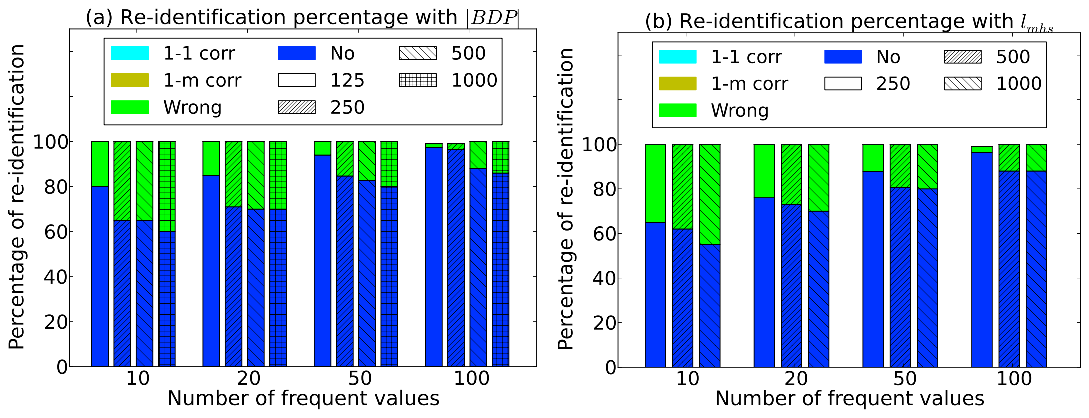


Figure 8.7: The cryptanalysis attack performed with different (a) number of block description pairs (BDPs) and (b) length of Min-Hash signatures (l_{mhs}).

8.5.3 Privacy

To evaluate the privacy of our approach we conducted the cryptanalysis attack (described in Section 4.5 on page 85) upon block representatives (B_{rep}s) because this is the only information a DO shares with another party about its generated blocks. We conducted this attack assuming an external attacker (such as the LU) colludes with a DO to learn about the parameter settings used in the B_{rep}s generation process. By using this information the attacker can regenerate the set of binary vectors ($\mathbf{V}_B$) that were used to generate the set of Min-Hash signatures (see Section 7.3 on page 141). Then, the attacker tries to re-identify a record in the database $\mathbf{D}$ of a non-colluding DO by using $\mathbf{V}_B$. We also assume the attacker uses a publicly available databases $\mathbb{G}$ which is a superset of the database $\mathbf{D}$ (i.e. $\mathbf{D} \subseteq \mathbb{G}$).

As illustrated in Figure 8.7 (a), we calculated the percentages of (1-to-1 correct, 1-to-m correct, wrong, and no) guesses with different number of block description pairs (BDPs). We noted that even with a set of 1,000 BDPs an attacker could not correctly re-identify an attribute value of a record in $\mathbf{D}$. We also conducted the cryptanalysis attack for different length of Min-Hash signatures (l_{msh}) as shown in Figure 8.7 (b). This figure shows that most of the re-identification guesses resulted in no guesses. This suggests that even when colluding with a DO an attacker still could not re-identify an attribute value of a record of a database $\mathbf{D}$ by using the representatives of the generated blocks of $\mathbf{D}$.

8.6 Summary

In this chapter we have proposed a novel scalable subgroup blocking technique that can be used in both classical and privacy-preserving multidatabase linkage scenarios. Based on the parameters minimum and maximum subgroup sizes, our approach can be used under different real-world scenarios. The approach uses a weighted graph structure for identifying similar blocks of different databases that need to be compared using a graph based candidate generation technique based on a depth-first based iterative deepening algorithm. The evaluation with several large real datasets indicated that our approach is scalable with the size of subgroups and number of databases and improves efficiency in candidate blocks generation.

As future work, we aim to improve the efficiency of our approach by adapting pattern growth methodologies [1]. One limitation we noted in the depth-first based candidate generation algorithm is that it requires a large number of recursive calls to generate candidate block tuples for increasing number of databases. Parallelisation of Algorithm 8.2 can help to overcome this limitation which is another future research avenue of our approach.

All the blocking techniques we have proposed so far in this thesis output a list of candidate block tuples that need to be compared later in the PPRL process by using an appropriate private comparison and classification technique. However, for a given subgroup of databases these candidate block tuples could potentially contain redundant record comparisons that result from repetitive and superfluous block comparisons. In the next chapter we describe a novel meta-blocking approach that improves the overall efficiency of the linkage process by removing such redundant record pair comparisons in the generated candidate block tuples.

Chapter 9

Scalable Meta-Blocking for Multidatabase Privacy-Preserving Record Linkage

As detailed in the previous chapters, blocking reduces the record comparison space by removing a large number of potential comparisons that correspond to non-matches, such that expensive similarity comparisons are only required on a smaller number of candidate record tuples. However, due to the increase in the number of databases, all multidatabase record linkage (MDRL) applications experience an exponential comparison space even after blocking. In this chapter we propose a novel scalable meta-blocking approach that can be used in both classical and privacy-preserving multidatabase record linkage (MD-PPRL) applications to significantly reduce the complexity of the comparison and classification step. In Section 9.1, we discuss the importance of removing unwanted record comparisons in a MD-PPRL application, and in Section 9.2, we provide an overview of our proposed approach. As detailed in Section 9.3, our approach utilises a graph data structure to schedule the comparison of pairs of blocks with the aim of minimising the number of unwanted comparisons between records. In Section 9.4, we provide a detailed analysis of our approach in terms of complexity, blocking quality, and privacy. In Section 9.5, we will describe the empirical study we conducted using large datasets, and finally we summarise our findings in Section 9.6.

9.1 Introduction

As discussed in Chapter 7, the use of distributed bocking in MDRL and MD-PPRL improves the overall efficiency in blocking since each database owner (DO) can perform blocking independently. However, even with distributed blocking, MD-PPRL still leads to a very large number of similarity comparisons because each record in a block needs to be compared with all records in the same block that originate from the other databases (that have the same or similar blocking key values, BKVs). Assuming d databases, each containing n_r records, split evenly into a set of blocks $\mathbf{B}$ of

Table 9.1: Notations and terminologies used in this chapter

$\mathbf{B}$	Set of blocks of a database owner
$\mathbf{BDP}$	List of BDPs
$\mathbf{CBT}$	List of candidate block tuples
$\mathbf{D}$	Database to be linked
$\mathbf{G}$, $\mathbf{G}_w$	Blocking graph and weighted blocking graph
$\mathbf{M}$	Set of matching blocks
B	A block
B_i, B_j	A block identifier
B_{rep}, B_{size}	A block representative and block size
BDP	List of block description pairs
CBT	A candidate block tuple
E	List of edges
M	Set of matching records
R, R_A, R_B	A record
V	List of vertices
b_{min}	Minimum block size
d	Number of database owners (or databases)
e	An edge in a graph
n_{CBT}, n_r	Number of CBTs and records
$sim()$	A similarity function
u, v	A vertex in a graph
w	Weight of an edge
BKV	Blocking key value
DO	Database owner
LSH	Locality sensitive hashing
LU	Linkage unit
MD-PPRL	Multidatabase privacy-preserving record linkage
RL	Record linkage

size $\frac{n_r}{|\mathbf{B}|}$ records each, will result in $|\mathbf{B}| \cdot (\frac{n_r}{|\mathbf{B}|})^d$ record pairs that need to be compared. Nevertheless, sending all databases to one party for linkage (such as a *linkage unit*, LU) does not reduce the above complexity since in a block each record needs to be compared with every other record. Therefore, some filtering techniques are required in between the blocking and comparison steps of MD-PPRL to reduce the record comparison space further.

To minimise the number of comparisons between records in MD-PPRL, we propose a novel graph based meta-blocking approach. Our approach is utilised in the third layer of our proposed blocking framework as explained in Chapter 4 on page 67. This meta-blocking step can be incorporated between the blocking and comparison steps in both the MDRL and MD-PPRL pipelines. As will be described in Section 9.3, our approach schedules pairs of blocks for comparison based on a list of candidate block tuples (CBTs). As shown in Figure 4.1 (on page 64), this list of CBTs can be generated either by the *block generation* or the *sub-group blocking* layers in our framework.

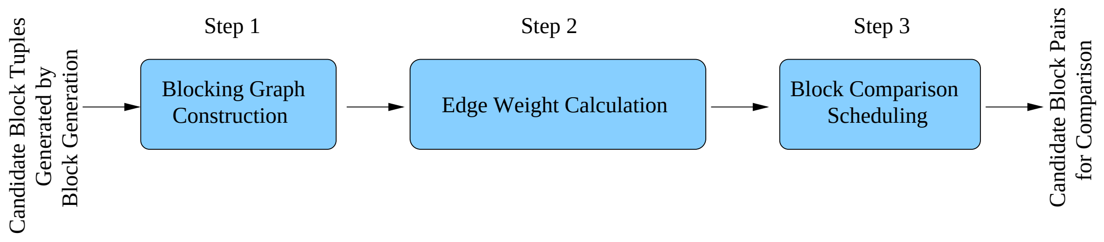

Step 1
Step 2
Step 3
Candidate Block Tuples Generated by Block Generation
Blocking Graph Construction
Edge Weight Calculation
Block Comparison Scheduling
Candidate Block Pairs for Comparison

Figure 9.1: An overview of our meta-blocking approach with its three main steps. As detailed in Section 9.3, the input to our approach is a list **CBT** of candidate block tuples. The *linkage unit* (LU) constructs a graph in step 1 using the **CBT**, where each block in a candidate block tuple ($CBT \in \mathbf{CBT}$) becomes a vertex, and two vertices are connected if they occur in the same CBT. In step 2, a normalised weight is calculated for each edge, while in step 3 block pairs in the **CBT** are sorted according to their edge weights to schedule pairs of blocks for comparison.

As we have reviewed in Chapter 3, several block processing and filtering techniques have been proposed for both RL and PPRL to reduce the record comparison space. The main drawback of these existing techniques is that their performance depends mainly on fine-tuning of application and data specific parameters. In contrast, our proposed approach does not require any data dependent parameters to be set. It aims to reduce the number of expensive record pair comparisons by discarding unwanted comparisons between records. Table 9.1 summarises the notation we use in this chapter and we next describe our approach in more details.

9.2 Overview of our approach

We assume d database owners (DOs), each with a database $\mathbf{D}$, are participating in our approach, with $d \geq 3$. We use the notation $\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C$, etc. to represent each database of DO_A, DO_B, DO_C, and so on. An overview of our proposed meta-blocking approach is shown in Figure 9.1. We employ a *linkage unit* (LU) to determine the ordering of pairs of blocks that need to be compared, and to coordinate the matching of blocks using appropriate comparison and classification techniques [35].

As detailed in Section 9.1 the main aim of our meta-blocking approach is to schedule pairs of blocks in a list of CBTs, **CBT**, to remove unwanted record pair comparisons to achieve high efficiency with no loss in effectiveness in the linkage process. As defined in Chapter 4 (on page 65) each candidate block tuple $CBT \in$ **CBT** contains identifiers of blocks that have the same or similar blocking key values (BKVs) from between 2 and d databases, and a maximum of one block identifier per database. We use the notation B_A, B_B, B_C, and B_D to represent identifiers of blocks of $\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C$, and $\mathbf{D}_D$, respectively. Figure 9.2 shows an illustrative example of our approach with a list **CBT** generated from these four databases.

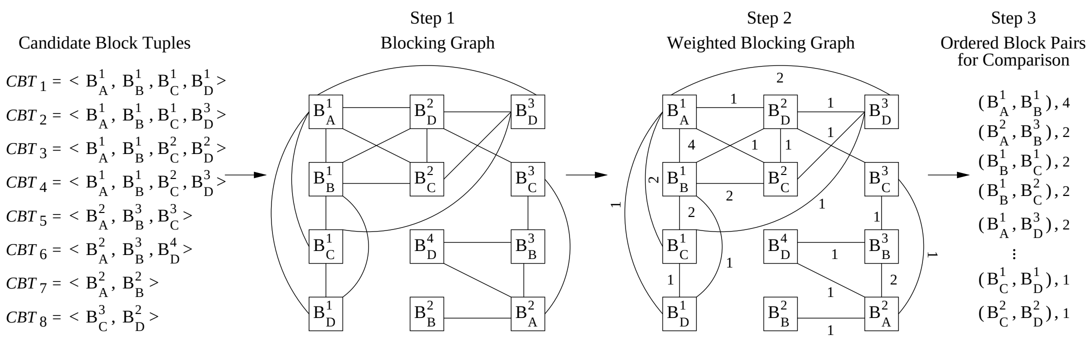

Candidate Block Tuples
$CBT_1 = < B^1_A, B^1_B, B^1_C, B^1_D >$
$CBT_2 = < B^1_A, B^1_B, B^1_C, B^3_D >$
$CBT_3 = < B^1_A, B^1_B, B^2_C, B^2_D >$
$CBT_4 = < B^1_A, B^1_B, B^2_C, B^3_D >$
$CBT_5 = < B^2_A, B^3_B, B^3_C >$
$CBT_6 = < B^2_A, B^3_B, B^4_D >$
$CBT_7 = < B^2_A, B^2_B >$
$CBT_8 = < B^3_C, B^2_D >$
Step 1
Blocking Graph
Step 2
Weighted Blocking Graph
Step 3
Ordered Block Pairs for Comparison
$(B^1_A, B^1_B), 4$
$(B^2_A, B^3_B), 2$
$(B^1_B, B^1_C), 2$
$(B^1_B, B^2_C), 2$
$(B^1_A, B^3_D), 2$
$\vdots$
$(B^1_C, B^1_D), 1$
$(B^2_C, B^2_D), 1$

Figure 9.2: An illustrative example of our meta-blocking approach. In this example the list of candidate block tuples (**CBT**) contains blocks from databases $\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C$, and $\mathbf{D}_D$. In step 1, these blocks are added to a blocking graph **G** as vertices and edges are created between these vertices. As describe in Section 9.3, each edge is assigned with a weight, in this example the *cardinality* (number of times a pair of blocks appears in **CBT**). In step 3, block pairs (edges) are ranked based on their weights for comparison. In this example the block pair (B^1_A, B^1_B) will be compared first because it has the highest edge weight (4) in the graph.

In our approach we consider two types of unwanted record comparisons to be removed which are (1) *repeated* and (2) *superfluous* record pair comparisons, as we define below. Repeated record pair comparisons occur when the same pair of blocks is compared repeatedly for multiple CBTs, while superfluous comparisons occur between records in a CBT where previously a non-match classification has been made, therefore rendering further comparisons with other records in the same CBT unnecessary as they cannot lead to a set of matching records across d databases.

The record pairs to be generated from the list **CBT** can comprise repeated and superfluous comparisons that affect the overall efficiency of a linkage protocol. Both constitute an excess of computation cost. We formally define repeated and superfluous comparisons as follows:

Definition 9.1. Repeated Comparison
If a record pair comparison (R_A, R_B) in a block pair (B_A, B_B), with $R_A \in B_A$, $R_B \in B_B$ and $(B_A, B_B) \in CBT_x$, has been compared, then (R_A, R_B) of $(B_A, B_B) \in \forall CBT_y \in \mathbf{CBT}$, and $CBT_x \neq CBT_y$, is considered as a *repeated comparison*.

Definition 9.2. Superfluous Comparison
By assuming a linkage of three databases $\mathbf{D}_A, \mathbf{D}_B$, and $\mathbf{D}_C$, a given $CBT_x = \langle B_A, B_B, B_C \rangle$, a record pair comparison (R_A, R_C) with $R_A \in B_A$, $R_C \in B_C$ is considered as a *superfluous comparison* if $\neg\exists\ R_B \in B_B$ that has been classified as a match with R_A in the block pair (B_A, B_B). Records R_A and R_C only need to be compared if both pairs (R_A, R_B) and (R_B, R_C) have been classified as matches in the two block pairs (B_A, B_B) and (B_B, B_C), respectively.

For example, in Figure 9.2 all record comparisons in block pair (B_A^2, B_B^3) would be repeated once (conducted twice) if we compare B_A^2 and B_B^3 for each CBT it occurs in, namely in $\langle B_A^2, B_B^3, B_C^3 \rangle$ and $\langle B_A^2, B_B^3, B_D^4 \rangle$. For $CBT_1 = \langle B_A^1, B_B^1, B_C^1, B_D^1 \rangle$ in Figure 9.2, a comparison of record $R_B \in B_B^1$ with any record $R_C \in B_C^1$ is considered as a superfluous comparison if R_B has already been classified as not matching with any record in B_A^1 in block pair comparison (B_A^1, B_B^1).

As illustrated in Figure 9.1, our approach accepts a **CBT** as input and outputs an ordering (ranking) of pairs of blocks that need to be compared in the matching (comparison and classification) step of PPRL. The **CBT** is used to construct an undirected graph where the block pairs in each CBT are scheduled for comparison based on the ordering of the calculated edge weights. As illustrated in Figure 9.1, our meta-blocking approach consists of three main steps:

1. *Blocking Graph Construction*: The LU constructs the blocking graph **G** based on the list **CBT** output by a previous layer in our framework. The blocks of each DO become vertices in **G** and edges are created between blocks in each CBT.

2. *Edge Weight Calculation*: The LU calculates a weight for each edge in **G**. These weights are used to rank the edges in the order of how they are to be processed. We propose five schemes for calculating edge weights in the generated graph which are described in more details in the following section.

3. *Block Comparison Scheduling*: Based on the calculated edge weights in **G**, the LU uses a block scheduling method to order the block pair comparisons in **CBT**. In the following section we will propose four block scheduling methods for ranking the block pair comparisons.

For example as shown in Figure 9.2, for $CBT_1 = \langle B_A^1, B_B^1, B_C^1, B_D^1 \rangle$ and $CBT_2 = \langle B_A^1, B_B^1, B_C^1, B_D^3 \rangle$, we aim to find an efficient ordering of the pairs of blocks in these two CBTs to be compared such that the ordering minimises the number of repeated and superfluous record pair comparisons. Two example possible orderings for these two CBTs are $(B_A^1, B_B^1) \rightarrow (B_B^1, B_C^1) \rightarrow (B_C^1, B_D^1) \rightarrow (B_C^1, B_D^3)$, or $(B_A^1, B_B^1) \rightarrow (B_B^1, B_C^1) \rightarrow (B_A^1, B_D^1) \rightarrow (B_A^1, B_D^3)$. We next describe our meta-blocking approach in more detail.

9.3 Scalable Meta-Blocking Approach

As illustrated in Figure 9.1, a list **CBT** of candidate block tuples that need to be compared across the databases in the comparison and classification step are input to our approach. We consider the generation of **CBT** to be a *black box* where CBTs can be generated by using any multidatabase blocking technique, such as those proposed in Chapters 5 to 8.

Apart from the list **CBT**, the LU requires additional information of a block to conduct the scheduling of block pairs. For every block in **CBT**, the LU requires a *block representative* (B_{rep}) and its *block size* (B_{size}). As described in Section 7.3 on page 136, a B_{rep} can be generated in different forms, for example as a representative

Algorithm 9.1: Blocking Graph Construction by the LU

```
Input  : CBT - List of candidate block tuples ($CBT$s)
         L   - List of pairs of ($B_{rep}$, $B_{size}$)s, where $\mathbf{L} \leftarrow [L_1, \cdots, L_d]$
Output: G    - Undirected blocking graph

$V \leftarrow \emptyset, E \leftarrow \emptyset$                         // Initialise an empty graph
foreach $CBT \in \mathbf{CBT}$ do                          // Iterate over every CBT
    foreach $B_i \in CBT$ do                            // Add each candidate block to graph
        $(B^i_{rep}, B^i_{size}) \leftarrow getBlockInfo(\mathbf{L}, B_i)$   // Get the corresponding pair of ($B_{rep}$, $B_{size}$)
        $v \leftarrow B_i$                                  // Create a new vertex
        $v.rep \leftarrow B^i_{rep}$               // Add the block representative as an attribute
        $v.size \leftarrow B^i_{size}$                    // Add the block size as an attribute
        $V.add(v)$                              // Add the newly created vertex to graph
        foreach $B_j \in CBT$ do                        // Add each candidate block to graph
            if $B_i \neq B_j \wedge (B_i, B_j) \neq E$ then          // Check if the edge already exists
                $E.add((B_i, B_j))$                          // Create a new edge
            end
        end
    end
end
return $\mathbf{G} = (V, E)$
```

Min-Hash signature [25, 101], a representative Bloom filter [190], or as a phonetic encoding [35]. B_{size} is the number of records in a block. The pairs of (B_{rep}, B_{size}) are used in step 2 of our approach to calculate the weights of each edge between blocks.

The technique used to generate B_{rep}s needs to be the same in all databases as the same type of B_{rep}s are required to allow the calculation of similarities between pairs of blocks that need to be compared across databases. Each DO$_i$ sends a list of pairs of (B_{rep}, B_{size}), L_i, to the LU for its corresponding blocks in **CBT**.

However, it is important to note that in a PPRL context revealing pairs of (B_{rep}, B_{size}) can increase the potential risk of frequency based cryptanalysis attacks as explained in Section 2.4 on page 33. We will analyse the privacy of each step of our meta-blocking approach within a PPRL context in Section 9.4. We next discuss each step of our approach in more detail.

1. **Blocking Graph Construction**

   As illustrated in Figure 9.1, in step 1 of our approach an undirected blocking graph **G** is created from the **CBT**. Similar to the candidate graph defined in Section 8.3 (on page 162), the blocks of different databases in **CBT** are represented as vertices in the blocking graph, however, the edges are created between these vertices only if their corresponding blocks appear in the same $CBT \in \mathbf{CBT}$. We formally define a blocking graph as follows.

   Definition 9.3. Blocking Graph **G**
   Given a list **CBT** of candidate block tuples, the undirected blocking graph derived from **CBT** is a d-partite graph $\mathbf{G} = (V, E)$, where V is a set of vertices and

E is a set of undirected edges that each is an unordered pair of elements of V. V represents all unique blocks in **CBT**, i.e. $V = \{\cup\, B_i \in CBT_j : \forall\, CBT_j \in \mathbf{CBT}\}$, while E contains all undirected edges between pairs of vetices $(B_i, B_j) \in V$ such that $E = \{(B_i, B_j) : (B_i \in CBT) \wedge (B_j \in CBT) \wedge (i \neq j), \forall\, CBT \in \mathbf{CBT}\}$.

The blocking graph construction is outlined in Algorithm 9.1. In line 2 the LU iterates through each CBT in **CBT**. Each B_i which represents a candidate block in a CBT is added as a new vertex v in **G** (lines 3 to 8). In line 4 the function *getBlockInfo()* returns the corresponding block description pair (B^i_{rep}, B^i_{size}) for each B_i. These are then added as attributes of v (lines 6 and 7). An edge is created between two vertices if a pair of B_is appears in a CBT, such that edges are created only between candidate blocks of different databases (lines 9 to 11).

2. **Edge Weight Calculation**

As shown in Figure 9.1, in the second step of our approach the blocking graph **G** is converted into an undirected weighted blocking graph $\mathbf{G}_w$ by calculating a normalised weight $w \in [0, 1]$ for each edge $e \in E$. Below we describe five weighting schemes to calculate edge weights, denoted by $e.w$, on characteristics drawn from the pair of vertices which created the edge. The general idea behind these weighting schemes is to recognise the block pairs' expected benefit of processing based on the blocks' characteristics. Block pairs with higher weights should be compared before the block pairs with lower weights to improve the efficiency of the overall linkage process.

- **Cardinality Based Weighting ($\mathbf{w_{card}}$)** This weighting scheme considers the importance of a pair of blocks in the list **CBT** as described by the cardinality of its corresponding edge in **G**. The cardinality of an edge (i.e. a block pair) is defined as the number of times the given pair of (B_i, B_j) appears in a $CBT \in \mathbf{CBT}$. The normalised weight w_{card} of edge $e_{(B_i,B_j)}$ is formally defined as:

$$w_{card}(B_i, B_j) = \frac{|\{(B_i, B_j) \in CBT : CBT \in \mathbf{CBT}\}|}{\max\limits_{\forall (B_i,B_j) \in E} (|\{(B_i, B_j) \in CBT, CBT \in \mathbf{CBT}\}|)}, \tag{9.1}$$

where $|\cdot|$ represents the cardinality of a given set and $max()$ returns the maximum cardinality of the set of edges in E. For example, in Figure 9.2 the block pair (B^1_A, B^1_B) has a cardinality of 4. Hence, the block pair (B^1_A, B^1_B) has a w_{card} of 1.0.

This scheme expresses the hypothesis that the more often a block pair occurs in the **CBT**, the more beneficial it would be to compare the pair at an early stage as this can improve the efficiency of the comparison step by reducing the number of repeated record pair comparisons in later block processing.

- **Comparison Based Weighting (w_{comp})**

  This scheme is based on the number of record pair comparisons that need to be performed between two blocks as calculated based on their block sizes. More formally, the normalised weight w_{comp} of edge (B_i, B_j) is defined as:

  $$w_{comp}(B_i, B_j) = 1 - \frac{B^i_{size} \cdot B^j_{size}}{max(\{B^i_{size} \cdot B^j_{size} \ : \ \forall(B_i, B_j) \in E\})}, \tag{9.2}$$

  where B^i_{size} and B^j_{size} are the size (number of records) of the vertices B_i and B_j, respectively, that connect the edge (B_i, B_j), and $max()$ returns the maximum number of record pair comparisons required between any pair of blocks in **CBT**.

  This scheme identifies the pairs of blocks that require a smaller number of record pair comparisons since an edge between the smallest blocks will receive the highest weight. Comparing smaller sized block pairs at an early stage will avoid superfluous record pair comparisons in later block comparisons.

- **Matching Based Weighting (w_{match})**

  In this scheme, we calculate a weight for each edge (B_i, B_j) between vertices B_i and B_j proportional to the cost of comparing the corresponding blocks (number of record pair comparisons) of the edge against the potential maximum number of matches that can be gained by comparing those two blocks (assuming the blocks contain no duplicate records). More formally, the normalised weight w_{match} of edge (B_i, B_j) is defined as:

  $$w_{match}(B_i, B_j) = \frac{min(B^i_{size}, B^j_{size})}{B^i_{size} \cdot B^j_{size}} = \frac{1}{max(B^i_{size}, B^j_{size})}. \tag{9.3}$$

  where $min()$ returns the minimum of two block sizes B^i_{size} and B^j_{size}.

  The hypothesis behind this matching based weighting scheme is to identify the block pairs that have a larger number of potential matches. Comparison of these block pairs at an early stage during block processing would effectively propagate the resulting matching record pairs into the neighbouring vertices in $\mathbf{G}_w$ thereby avoid superfluous comparisons because non-matching records will not be compared with the records in corresponding blocks of neighbouring vertices.

- **Similarity Based Weighting (w_{sim})**

  In this scheme, the similarity between block representatives (B_{rep}s) of a pair of blocks is used as the weight of an edge connecting those two blocks. We assume B_{rep}s (that depend upon the actual blocking technique used) can be compared using appropriate similarity functions to measure their similarity. For example, the Jaccard coefficient [35], Dice coefficient [216], and edit-distance [35] can be used with Min-Hash signature, Bloom filter, and phonetic encoding representatives, respectively. Hence, the normalised weight w_{sim} of edge (B_i, B_j) is defined as:

$$w_{sim}(B_i, B_j) = sim(B^i_{rep}, B^j_{rep}), \tag{9.4}$$

where B^i_{rep} and B^j_{rep} are the block representatives of vertices B_i and B_j, respectively, and $sim()$ is an appropriate similarity function that calculates the similarity between B^i_{rep} and B^j_{rep}, with $0 \leq sim() \leq 1$.

This scheme is based on the hypothesis that the more similar two block representatives are, the more likely the corresponding blocks contain more matching record pairs. Comparing more similar blocks at an early stage enables the identification of a set of matching records that reduces the number of superfluous record comparisons in later block comparisons.

- **Combined Weighting (w_{comb})**

  This weighting scheme is a combination of the previous four schemes, where each scheme is given a weight value to differentiate the importance of each scheme in the overall edge weight. For each edge $(B_i, B_j) \in E$ in the graph the weight w_{comb} is calculated as a generic score which is formally defined as (with $\alpha, \beta, \delta, \gamma \geq 0$):

$$\begin{gathered} w = \alpha \cdot w_{card}(B_i, B_j) + \beta \cdot w_{comp}(B_i, B_j) + \delta \cdot w_{match}(B_i, B_j) + \gamma \cdot w_{sim}(B_i, B_j) \\ w_{comb}(B_i, B_j) = \frac{w}{\alpha + \beta + \delta + \gamma}, \; where \; (\alpha + \beta + \delta + \gamma) > 0. \end{gathered} \tag{9.5}$$

3. **Block Comparison Scheduling**

   The third step of our approach aims to rank (order) the edges E in the graph $\mathbf{G}_w$ according to their weights, as shown in Figure 9.2. This ranking provides an order for the comparison of block pairs in the **CBT** that allows for a significant reduction in the overall number of repeated and superfluous record pair comparisons during the comparison and classification step in MDRL or MD-PPRL pipeline.

   For the scheduling of block comparisons we consider two aspects of: (1) propagation of previous comparison results to subsequent block comparisons, and (2) distribution of block processing. The first aspect consists of the two categories *static* and *dynamic* based upon if the blocking graph with the comparison results is updated or not during the scheduling process. In the static approach, the set of edges E is ranked only once and the graph $\mathbf{G}_w$ is not updated with any results from the conducted block pair comparisons. In the dynamic approach the comparison results of the previous block pair comparisons are added to $\mathbf{G}_w$ as new nodes and E is re-ranked after each compared block pair.

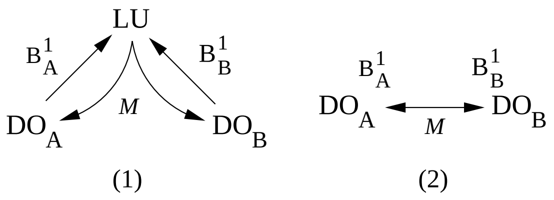


Figure 9.3: An example of generating the block of matching records M for block pair (B^1_A, B^1_B) under the (1) *centralised* and (2) *distributed* match settings. In the centralised setting a three party comparison and classification (matching) technique is used where the database owners (DO$_A$ and DO$_B$) send their blocks to the *linkage unit* (LU) to compare the blocks. In the distributed setting the DOs participate in a two party matching protocol to generate M. Arrows represent the communication between parties.

In the second aspect we consider how the actual matching of block pairs is conducted, categorised into *centralised* and *distributed*. An example of the comparison of block pair (B^1_A, B^1_B) under these two categories is shown in Figure 9.3. In the centralised setting we assume all the blocks are compared by the LU or by another external party using existing comparison and classification techniques [215] based on the blocks sent to it by the DOs. In the distributed category we assume the DOs compare blocks among themselves without any involvement of a third party.

Based on these two aspects we propose four block comparison scheduling methods as described below. We assume a function *matchBlocks()* which conducts the comparison and classification for a given pair of blocks, and which returns a set of matching records (M) [15].

- **Static Centralised Block Scheduling**

  Algorithm 9.2 outlines the main steps involved in the static centralised block comparison scheduling method. This approach can be used in a scenario where the LU [174] performs the record comparisons between the blocks that have been sent to it by the DOs. Block pairs are processed sequentially and the graph $\mathbf{G}_w$ is not updated after a pair of blocks has been compared.

  In lines 2 and 3 of Algorithm 9.2, the edges in $\mathbf{G}_w$ are ranked in descending order according to their weights. Once a block pair is compared (line 5), the set of records that are classified as matches M is added to a list V_c of matched records (line 6) for later comparisons [15]. To avoid any superfluous comparisons in later block pair comparisons in the CBTs, the set of matches M is used to update the corresponding block pair comparisons accordingly in the relevant CBTs (lines 7 to 9).

  For example, once compared, the block pair comparison (B^1_A, B^1_B) is updated in CBT_1 and CBT_2 given in Figure 9.2, with M as $\langle M, B^1_C, B^1_D\rangle$ and $\langle M, B^1_C, B^3_D\rangle$, respectively. This ensures that previously compared block

Algorithm 9.2: Static Centralised Block Scheduling

```
Input  : $\mathbf{G}_w$  - Undirected weighted blocking graph
         CBT - List of candidate block tuples ($CBT$s)
Output: M    - Matching records sets
1  $V_c \leftarrow [], \mathbf{M} \leftarrow \emptyset$                         // Initialise the variables
2  $E \leftarrow getEdges(\mathbf{G}_w)$                        // Get set of edges from $\mathbf{G}_w$
3  $E_r \leftarrow rankEdges(E)$                                 // Rank the set of edges
4  foreach $(B_i, B_j) \in E_r$ do                         // Iterate over the ranked edges
5      $M \leftarrow matchBlocks(B_i, B_j, [])$                     // Compare the two blocks
6      $V_c.add(M)$                            // Add the set of matched records to $V_c$
7      foreach $CBT \in$ CBT do                                // Iterate over the $CBT$s
8          $CBT \leftarrow updateCandidateSet(CBT, M)$     // Update the corresponding $CBT$s
9      end
10 end
11 foreach $CBT \in$ CBT do                                    // Iterate over the $CBT$s
12     foreach $(B_i, B_j) \in CBT$ do                   // Compare the remaining blocks
13         $M \leftarrow matchBlocks(B_i, B_j, V_c)$                      // Compare the blocks
14         $\mathbf{M} \leftarrow \mathbf{M} \cup M$                  // Add to final matched record set
15     end
16 end
17 return M
```

pairs are not compared repeatedly in later block pair comparisons of any $CBT \in$ **CBT**. Finally, the remaining block pairs in V_c, which are not updated in $\mathbf{G}_w$, are compared (lines 11 to 16) and the set of matching records are added to the final set of matching records **M** in line 14.

- **Dynamic Centralised Block Scheduling**

  Similar to the static centralised scheduling method, the dynamic centralised block comparison scheduling method also processes each edge (B_i, B_j) between vertices u and v in $\mathbf{G}_w$ sequentially, but $\mathbf{G}_w$ is updated according to the results obtained from comparing the block pair of (B_i, B_j) (i.e., set of compared records that are classified as matches M). As outlined in Algorithm 9.3, this method iterates through the set of edges E in $\mathbf{G}_w$ (line 2) where at a given iteration the edge (B_i, B_j) with the highest weight is identified for comparison by the function *getTopWeightEdge()* (line 3).

  In line 4 in Algorithm 9.3, the set M of matching records between the two compared blocks is generated. The processed edge (B_i, B_j) is examined for any adjacent (neighbouring) nodes which share the same CBT (line 5). If the processed edge (B_i, B_j) is the last block pair in a given CBT then M is added to the final set of matches **M** (lines 13 and 14).

  If the edge (B_i, B_j) contains neighbouring nodes, $\mathbf{G}_w$ needs to be updated with the compound block of matches M (lines 5 to 12). To represent M in $\mathbf{G}_w$ a new vertex (let us assume as v') is created by generating a new block identifier and the respective attribute values (size and block repre-

Algorithm 9.3: Dynamic Centralised Block Scheduling

```
Input  : G_w - Undirected weighted blocking graph
         ws  - Weighting scheme
Output: M   - Matching records sets
1  $\mathbf{M} \leftarrow \emptyset$                                   // Initialise an empty set of matching records
2  while $\mathbf{G}_w.E \neq \emptyset$ do                          // Iterate through all the edges in $\mathbf{G}_w$
3      $(B_i, B_j) \leftarrow getTopWeightEdge(\mathbf{G}_w.E)$          // Get the highest weighted edge
4      $M \leftarrow matchBlocks(B_i, B_j, [])$                         // Compare the two blocks
5      if $(B_i, B_j).neighbours() \neq \emptyset$ then               // Check for neighbouring vertices
6          $\mathbf{G}_w.V \leftarrow createVertex(M)$                    // Create a new vertex
7          foreach $v \in (B_i, B_j).neighbours()$ do                 // Iterate over each neighbour
8              $\mathbf{G}_w.E \leftarrow createEdge(v, (B_i, B_j))$      // Create a new edge
9          end
10         $\mathbf{G}_w.E \leftarrow updateWeights(\mathbf{G}_w.E, ws)$    // Recalculate the edge weightes
11         $\mathbf{G}_w.E \leftarrow removeEdges((B_i, B_j), (B_i, B_j).neighbours())$   // remove unwanted edges
12     end
13     else
14         $\mathbf{M} \leftarrow \mathbf{M} \cup M$                     // Add to final matched record set
15     end
16     $\mathbf{G}_w.V \leftarrow removeVertices(B_i, B_j, \mathbf{G}_w.V, \mathbf{G}_w.E)$   // Remove compred vertices from G
17 end
18 return M
```

sentative) of v' are added based the characteristics of M (line 6). $v'.size$ is updated with the number of records in M as the block size ($B_{size} = |M|$), however, the attribute $v'.rep$ is excluded because the LU does not know the parameter settings used by the DOs in the B_{rep} generation.

Since v' does not contain a B_{rep} for later similarity calculations for the similarity based (w_{sim}) and combined (w_{comb}) weighting schemes we use the *average similarity*. The average similarity is computed by taking the average of similarities between the B_{rep}s of B_i and B_j, with the B_{rep} of neighbouring vertex.

Once the new vertex v' is added into $\mathbf{G}_w$, new edges are created between each neighbouring vertices of (B_i, B_j) and v' (lines 7 and 8). This ensures that superfluous comparisons are not included in the later block comparisons of the neighbouring edges that share the same CBT. The weights of these new edges are recalculated (line 10) according to the edge weighting scheme ws that is being used in $\mathbf{G}_w$ (as discussed for step 2 above).

The set of edges that connects the neighbours of (B_i, B_j) with vertices B_i and B_j are removed from $\mathbf{G}_w$ by the function *removeEdges()* (line 11). To avoid repeated comparisons, this function also removes the edge (B_i, B_j) to ensure that the corresponding block pair will not be compared in any later CBTs. Finally, the function *removeVertices()* removes vertices B_i and B_j from $\mathbf{G}_w$ only if they are not connected with any other vertices (line 16).

Algorithm 9.4: Static Distributed Block Scheduling

```
Input  : $\mathbf{G}_w$  - Undirected weighted blocking graph
         CBT - List of candidate block tuples ($CBT$s)
         $d$   - Number of database owners (DOs)
Output: M    - Matching records sets

$V_c \leftarrow []$, $\mathbf{M} \leftarrow \emptyset$                      // Initialise the variables
$E \leftarrow getEdges(\mathbf{G}_w)$                      // Get set of edges from $\mathbf{G}_w$
$\mathbf{I} \leftarrow getEdgeIndex(E, d)$                 // Add edges into an inverted index
foreach $k \in |\mathbf{I}|$ do                          // Iterate over the edge index
    foreach $(B_i, B_j) \in \mathbf{I}[k]$ do            // Iterate over each edge in $\mathbf{I}[k]$
        $M \leftarrow matchBlocks(B_i, B_j, [])$         // Compare the two blocks
        $V_c.add(M)$                          // Add the set of matched records to $V_c$
        foreach $CBT \in$ CBT do                      // Iterate over the $CBT$s
            if $B_i \in CBT \wedge B_j \in CBT$ then     // Check if both blocks are in a $CBT$
                $CBT \leftarrow updateCandidateSet(CBT, M)$   // Update corresponding $CBT$s
            end
        end
    end
end
foreach $CBT \in$ CBT do                              // Iterate over the $CBT$s
    foreach $(B_i, B_j) \in CBT$ do                  // Compare the remaining blocks
        $M \leftarrow matchBlocks(B_i, B_j, V_c)$            // Compare the blocks
        $\mathbf{M} \leftarrow \mathbf{M} \cup M$                       // Add to final matched record set
    end
end
return M
```

- **Static Distributed Block Scheduling**

  The aim of this block scheduling approach is to order pairs of blocks such that they can be compared in a distributed manner across DOs without any involvement of a LU, as outlined in Algorithm 9.4. In Figure 9.2, for example, the block pair (B^1_A, B^1_B) can be compared by the owners of $\mathbf{D}_A$ and $\mathbf{D}_B$, and independently (B^1_C, B^1_D) by the owners of $\mathbf{D}_C$ and $\mathbf{D}_D$.

  In Algorithm 9.4, $\mathbf{G}_w$, **CBT**, and the number of DOs d are provided as inputs. The set of edges E in graph $\mathbf{G}_w$ is added into an inverted index $\mathbf{I}$ by the function *getEdgeIndex()* (lines 1 to 3). Each list $\mathbf{I}[k]$ holds the d edges that can be processed independently in a given iteration. These edges are selected considering the adjacent vertices (neighbours) and *CBT*s that include these pairs of blocks. Any block pairs that share a common neighbour, or if any of their corresponding blocks appears in the same *CBT*, are not added to the same list in $\mathbf{I}$. In line 6, each block pair in $\mathbf{I}[k]$ is sent to the relevant DOs to compare the records in the corresponding blocks.

  Similar to Algorithm 9.2, **CBT** is updated (lines 8 to 12) to avoid any repeated and superfluous comparisons. Compared to Algorithm 9.2, however, Algorithm 9.4 enables block pairs to be compared independently.

Algorithm 9.5: Dynamic Distributed Block Scheduling

```
Input  : $\mathbf{G}_w$ - Undirected weighted blocking graph
         $ws$ - Weighting scheme
         $d$ - Number of database owners (DOs)
Output: $\mathbf{M}$ - Matching records sets
$V_c \leftarrow [], \mathbf{M} \leftarrow \emptyset$                       // Initialise the variables
while $\mathbf{G}_w.E \neq \emptyset$ do                          // Iterate over all the edges in $\mathbf{G}_w$
    $E_d \leftarrow getTopWeightEdges(\mathbf{G}_w.E, d)$          // Get $d$ highest weighted edges
    foreach $(B_i, B_j) \in E_d$ do                        // Iterate over selected edges
        $M \leftarrow matchBlocks(B_i, B_j, [])$                 // Compare the two blocks
        if $(B_i, B_j).neighbours() \neq \emptyset$ then         // Check for neighbouring vertices
            $V_c.add(M)$                                  // Add the set of matched records to $V_c$
        end
        else
            $\mathbf{M} \leftarrow \mathbf{M} \cup M$                          // Add to final matched record set
        end
    end
    if $V_c \neq []$ then                        // Check if there are $M$s that has neighbours
        $\mathbf{G}_w \leftarrow addVerticesEdges(\mathbf{G}_w, V_c, E_d)$       // Create new vertices and edges
        $\mathbf{G}_w.E \leftarrow updateWeights(\mathbf{G}_w.E, ws)$          // Recalculate the edge weightes
        foreach $(B_i, B_j) \in E_d$ do                  // Iterate over each compared edges
            $\mathbf{G}_w.V \leftarrow removeVertices(B_i, B_j, \mathbf{G}_w.V, \mathbf{G}_w.E)$   // Remove compared vertices
        end
    end
end
return $\mathbf{M}$
```

- **Dynamic Distributed Block Scheduling**

  Similar to dynamic centralised scheduling, dynamic distributed scheduling updates the graph $\mathbf{G}_w$ according to record pair comparisons performed while the corresponding blocks of each edge are processed distributively among the DOs.

  As outlined in Algorithm 9.5, this method iterates through the set of edges in $\mathbf{G}_w$ (line 2). In line 3, the function *getTopWeightEdges()* ranks the set of edges $\mathbf{G}_w.E$ according to their weights and identifies the set of edges E_d that can be compared independently across the DOs. At a given iteration, E_d contains upto d edges (the number of databases to be linked). Similar to Algorithm 9.4, each block pair in E_d is assigned to the relevant DOs to perform the comparison and classification of the record pairs formed from these two blocks (lines 4 and 5).

  Similar to Algorithm 9.3, the blocking graph $\mathbf{G}_w$ is updated accordingly with the set of matching records M (lines 13 to 19). If a compared edge (B_i, B_j) contains neighbouring vertices then a new vertex is created for M and new edges are created appropriately with the neighbouring vertices (line 14) otherwise M is added to the final matching record set $\mathbf{M}$ (line 10).

Depending upon the weighting scheme *ws* used, the weights are calculated for the newly created edges by the function *updateWeights()* (line 15). To avoid repeated comparisons, the function *removeVertices()* then removes vertices B_i and B_j (line 17). This ensures the corresponding pair of blocks of B_i and B_j is not compared in any later CBTs. Updating $\mathbf{G}_w$ ensures that any superfluous comparisons in the neighbouring edges and repeated block pair comparisons will be avoided in later comparisons.

9.4 Conceptual Analysis of Meta Blocking Approach

We now analyse our meta-blocking approach in terms of complexity, blocking quality, and privacy assuming a MD-PPRL context. We assume the comparison and classification techniques used in our scheduling methods are accurate and provide enough privacy against possible privacy attacks.

9.4.1 Complexity

Let us assume $d \geq 3$ DOs participate in our approach with each having a database $\mathbf{D}$ to be linked. We assume there are $n_{CBT} = |\mathbf{CBT}|$ candidate block tuples (CBT) to be compared. Each CBT consists of d blocks, one each for the d databases. We do not consider the complexities of the CBT generation and private comparison and classification technique used since they are outside of our meta-blocking approach.

In step 1 of our approach, the n_{CBT} CBTs are added to the graph $\mathbf{G}$ as outlined in Algorithm 9.1. This requires the LU to iterate through each $B_i \in CBT$ to add as a vertex v to $\mathbf{G}$, and to create edges between vertices if they share the same CBT. A CBT with d candidate blocks generates $\frac{d(d-1)}{2}$ edges in $\mathbf{G}$. Hence, the construction of $\mathbf{G}$ is of complexity of $O(|E|)$, where $|E| = n_{CBT} \cdot d^2$.

In step 2 of the approach, weights need to be calculated for each edge in the graph $\mathbf{G}$ which requires the LU to iterate through the set of edges E in $\mathbf{G}$. This process has a computational complexity of $O(|E|)$.

In step 3, we assume the sorting algorithm used in the ranking function has a complexity of $O(|E| \cdot log(|E|))$. At a given iteration both centralised block scheduling methods select an edge for processing. Static centralised block scheduling has a complexity of $O(|E| \cdot log(|E|) + |E|)$ since it does not update $\mathbf{G}_w$ after a pair of blocks has been compared. On the other hand, dynamic centralised block scheduling has a complexity of $O(|E|^2 \cdot log(|E|) + |V| \cdot |E|)$ as it updates $\mathbf{G}_w$ after each comparison of a block pair.

The two distributed block scheduling methods select upto d edges for comparing block pairs in each iteration. This results in static distributed block scheduling to have a complexity of $O(|E| \cdot log(|E|) + |E| \cdot n_{CBT} \cdot d)$, while dynamic distributed block scheduling has a complexity of $O(|E|^2 \cdot log(|E|) + n_{CBT} \cdot d + |V| \cdot |E|)$ because $\mathbf{G}_w$ needs to be updated after pairs of blocks have been compared independently by the DOs.

9.4.2 Blocking Quality

Our meta-blocking approach ensures that all unique block pairs in the input **CBT** are considered in the block scheduling step. This means no block pairs are excluded for comparisons. Therefore, the number of true matching record sets in the **CBT** given as input to our approach remains the same as the number of true matching record sets output by our approach. This leads to no loss in effectiveness as no true matches are excluded for comparison by our approach.

In step 2, the weighting scheme applied on the graph $\mathbf{G}_w$ influences the reduction in the number of repeated and superfluous record pair comparisons. Block pair scheduling using the cardinality (w_{card}) based weighting scheme can reduce more repeated comparisons than the other schemes since block pairs whose edges have a high cardinality are compared first. On the other hand, using the comparison (w_{comp}) based weighting scheme leads to the comparison of block pairs with the lowest number of comparisons first, which can reduce more superfluous comparisons compared to the other schemes.

The matching (w_{match}) and similarity (w_{sim}) based weighting schemes calculate edge weights based on the number of potential matches and the similarity between block representatives, respectively. The reduction of record pair comparisons when using one of these two schemes, therefore, depends upon the blocking, comparison and classification techniques used.

In step 3 of our approach, compared to static block scheduling, the dynamic block scheduling methods update $\mathbf{G}_w$ with the resulting set of matching records once a pair of blocks has been compared. This ensures that comparison results are propagated to subsequent block comparisons to obtain the edge in $\mathbf{G}_w$ with the highest weight for the next comparison, leading to an efficient reduction in the number of record pair comparisons. In the distributed block scheduling methods, block pairs in the list of edges are compared independently which ensures $\mathbf{G}_w$ is processed more efficiently compared to when using the centralised block scheduling methods.

9.4.3 Privacy

We assume each party that participates in our approach follows the honest-but-curious (HBC) adversary model [212], as described in Section 2.4 on page 27. We assume each DO generates its blocks independently by using a private blocking technique as described in Chapter 7. Besides agreement of parameters, this technique does not require any further communication between the DOs that would reveal information about their blocks. We also assume each block contains at least $k = \frac{n_r}{|\mathbf{B}|}$ records to ensure k-anonymous privacy [200]. Therefore, no DO can learn anything about any other DO's sensitive data during the blocking step.

Once the blocks are generated, the LU could use the techniques described in Chapters 7 or 8 to generate the list of candidate block tuples (CBTs) for comparison. As we described in Section 8.4 on page 167, the LU cannot learn about the individual records in the CBT generation process. Apart from the block identifiers each DO needs to send the block representatives (B_{rep}s) and sizes (B_{size}s) of its blocks to the

LU. However, as we have shown in Section 8.5 (on page 168), the LU cannot re-identify a record in a block even if a DO reveals the parameter settings of the B_{rep} generation process to the LU.

The centralised scheduling methods presented in Section 9.3 require a third-party to conduct the private comparison and classification step [56, 176]. Either the LU or a separate external party is engaged to perform the linkage of masked records from individual pairs of blocks. We assume such a private comparison and classification protocol is available, where during the block pair comparison process no information about sensitive data is leaked between the DOs and the LU, or to any other party [216]. We do not analyse the privacy risk involved in this private comparison and classification protocol since it is not included as a step in our meta-blocking approach. However, as we described in Section 8.4, the LU is not capable of performing a cryptanalysis attack on the matching results of the block pair comparisons even if it conducts the linkage of pairs of blocks sent to it by the DOs, because an individual block does not contain enough frequency information to conduct such an attack. Furthermore, it has been shown in the past [132], even sophisticated cryptanalysis attacks based on a constrained satisfaction solver that learns information about some individual records in a block are only successful for certain parameter settings of the encoding methods used.

In the static and dynamic distributed scheduling techniques each block pair is compared using a two-party private comparison and classification technique without involvement of a LU. For the private comparison and classification step, for the distributed scheduling methods presented in Section 9.3, we assume the private matching protocol used is secure and no sensitive information is leaked to the individual DOs about each others' data.

In the dynamic distributed scheduling method, once a block pair is compared the blocking graph $\mathbf{G}_w$ needs to be updated with the information of the resulting set of match records M. This requires the DOs to send the size of M to the LU which reveals information about the frequency distribution of the blocks. However, as we have described above, since the LU does not know anything about the block representative generation and the masking of the records it becomes harder for the LU to infer information about individual records in a block.

9.5 Experimental Evaluation

In this section, we present and discuss the results of the experimental evaluation study of our meta-blocking approach conducted on the datasets described in Section 4.4 (on page 78). We name the four different block comparison scheduling methods as SCS for static centralised scheduling, DCS for dynamic centralised scheduling, SDS for static distributed scheduling, and DDS for dynamic distributed scheduling, respectively.

We use the same initial steps and parameter settings described in Chapter 8 to generate the set of candidate block tuples (CBTs). To evaluate how many record pairs

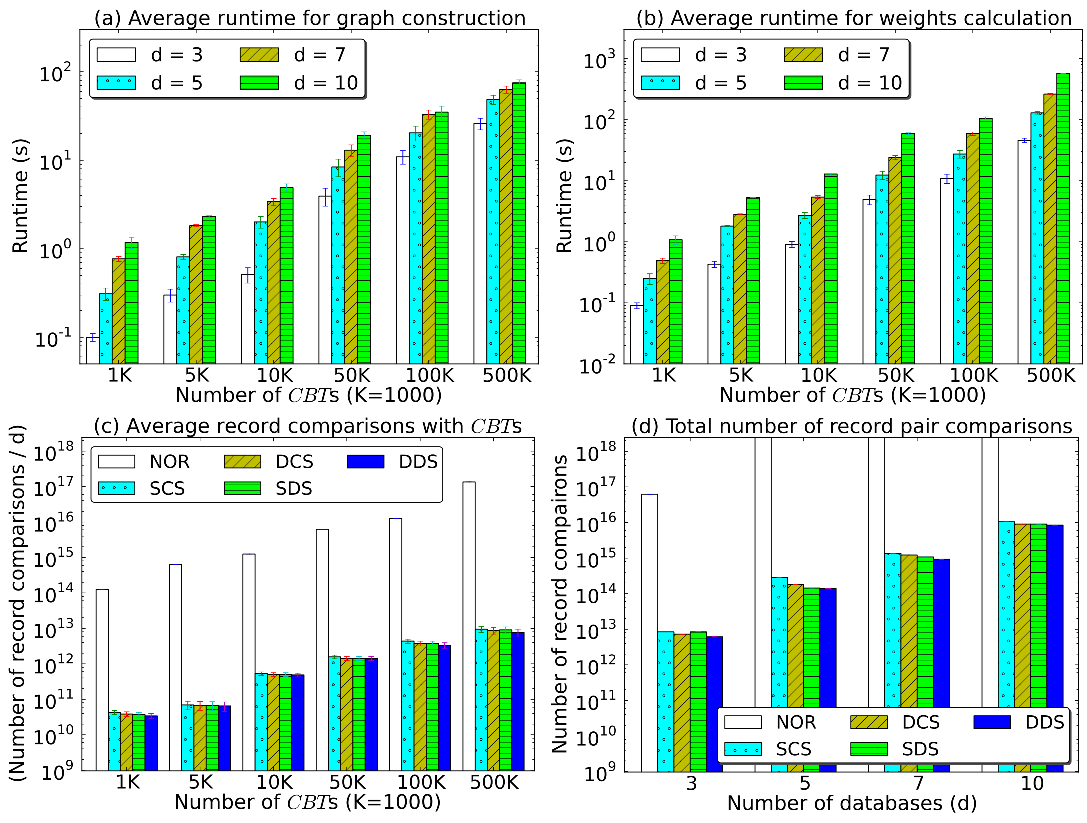


Figure 9.4: Average runtime for (a) blocking graph generation and (b) edge weight calculation with different number of candidate block tuples (*CBT*s). (c) shows the average record pair comparisons and (d) shows the total record comparisons required for different number of databases with 500K *CBT*s (the total number of comparisons required for the NOR approach for 5, 7 and 10 databases is higher than 10^{18}, and therefore not visible). Note that plots have different y-axis scales.

Table 9.2: Parameter settings used in the experimental evaluation

Parameter	Value range
Dataset name	NC-SYN
Number of databases (d)	3, 5, 7, 10
Number of records in each database (n_R)	1,000,000
Blocking key attributes (A)	Given name, Surname, City (suburb), and Postcode
Corruption levels	0%, 20%, and 40%
Number of candidate block tuples (n_{CBT})	1,000 to 500,000

are reduced by our approach we use block pair comparison with normal scheduling (NOR) as a baseline approach. In the NOR approach block pairs of a *CBT* are compared sequentially where each record in a block is compared with every record in all other blocks of a *CBT*. For example, the block pairs of $CBT_1 = \langle B^1_A, B^1_B, B^1_C, B^1_D \rangle$ in Figure 9.2 is compared as $(B^1_A, B^1_B) \rightarrow (B^1_B, B^1_C) \rightarrow (B^1_C, B^1_D)$. Table 9.2 summarises the parameter values used in our approach.

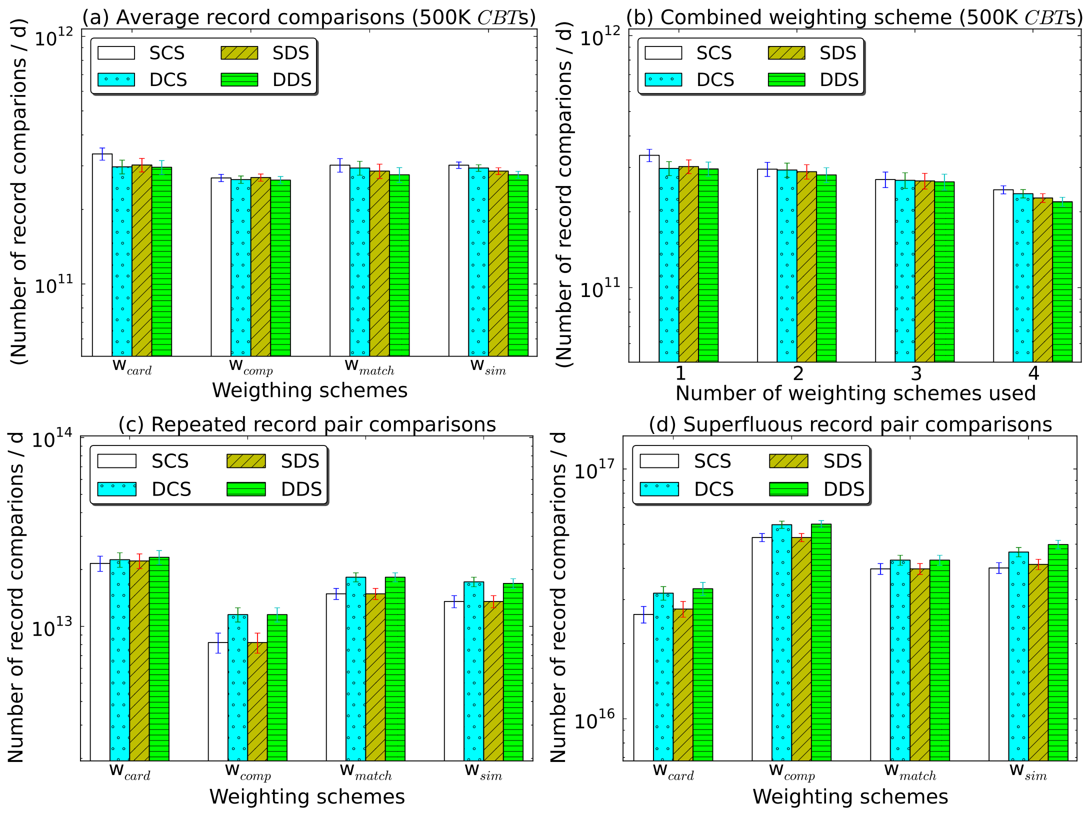


Figure 9.5: The average number of record pair comparisons required for (a) different weighting schemes and (b) for different weighting scheme combinations for different scheduling methods on 500K *CBT*s. Reduction in the number of (c) repeated and (d) superfluous comparisons achieved by different scheduling methods on 500K *CBT*s. Note that plots have different y-axis scales.

9.5.1 Scalability

Figures 9.4 (a) and (b) show the average runtime required for the blocking graph (**G**) construction and edge weight calculation steps of our approach. The runtime required for constructing **G** increases linearly with the number of candidate block tuples (n_{CBT}). We noted that the average runtime increases with the number of databases (d) as more blocks are added to **G** as vertices and $d(d-1)/2$ edges are created for each CBT. As we expected the runtime for the edge weight calculation step increases with n_{CBT} as weights need to be calculated for more edges in **G**.

Figure 9.4 (c) shows the scalability of our approach in terms of average record comparisons required with n_{CBT} and d. As can be seen, the average number of record comparisons required by each scheduling method scales linearly with n_{CBT}. NOR required more record comparisons because records are compared across all the blocks in a CBT. Figure 9.4 (d) shows the total number of record comparisons required by each scheduling method scales linearly with d. However, the number of record comparisons increases with d as more block pairs need to be processed.

Figure 9.5 (a) illustrates the average number of record pair comparisons performed per database with different weighting schemes and scheduling methods. As expected the dynamic distributed scheduling (DDS) method leads to a higher reduction in record pair comparisons compared to the other scheduling methods. This is also illustrated in Figures 9.4 (c) and (d) as DDS compares block pairs completely distributed leading to an improvement in the overall efficiency of the protocol.

As shown in Figure 9.5 (a), compared to the cardinality (w_{card}), matching (w_{match}), and similarity (w_{sim}) based weighting schemes, the comparison (w_{comp}) based weighting scheme performs better as it is capable of removing more record pair comparisons. This happens because w_{comp} selects the block pairs with the smallest number of record pair comparisons first and therefore removes more non matching record pairs at an early stage of the comparison process.

Figure 9.5 (b) shows the number of record pair comparisons required by each block scheduling method with the combination based weighting (w_{comb}) scheme. We set the constants α, β, δ, and γ in w_{comb} to either 0 or 1 such that $1 \leq \alpha + \beta + \delta + \gamma \leq 4$, and we measure the average number of record pair comparisons per database for different such combinations. The results show that the number of comparisons can be reduced better when several weighting schemes are combined.

Figures 9.5 (c) and (d) illustrate the number of repeated and superfluous record comparisons removed by each scheduling method with 500,000 CBTs for different weighting schemes. In Figure 9.5 (c), each scheduling method removes more repeated comparisons when used with w_{card} weighting compared to the other weighting schemes. This is because with w_{card} a block pair that is shared by multiple CBTs is compared first, and before block pairs that only occur once in a CBT. w_{card} also helps to shrink the block graph **G** by selecting the edges with highest cardinality which indirectly improves the overall efficiency of the linkage process.

As shown in Figure 9.5 (d), with the w_{comp} scheme more superfluous record pair comparisons are removed by all scheduling methods compared to the other schemes. As shown in these figures, while w_{card} and w_{comp} are better capable to either reduce repeated or superfluous comparisons (but not both), w_{match} and w_{sim} can reduce both types however less efficiently than w_{card} and w_{comp}. Overall, the dynamic scheduling methods perform more effectively than the static approaches, and the distributed approaches compare block pairs more efficiently than the centralised approaches.

9.5.2 Blocking Quality

Figure 9.6 (a) illustrates the effectiveness of our meta-blocking approach in terms of pairs completeness (PC) and reduction ratio (RR) measured for different block scheduling methods for 500,000 CBTs. This figure shows scheduling does not reduce the number of true matches in the compared pairs of blocks, while each scheduling method increases the RR considerably compared to the NOR baseline. Figure 9.6 (b) shows the use of scheduling can improve the overall effectiveness of linkage even for large numbers of CBTs compared to NOR which leads to lower FM values due to its large number of record comparisons.

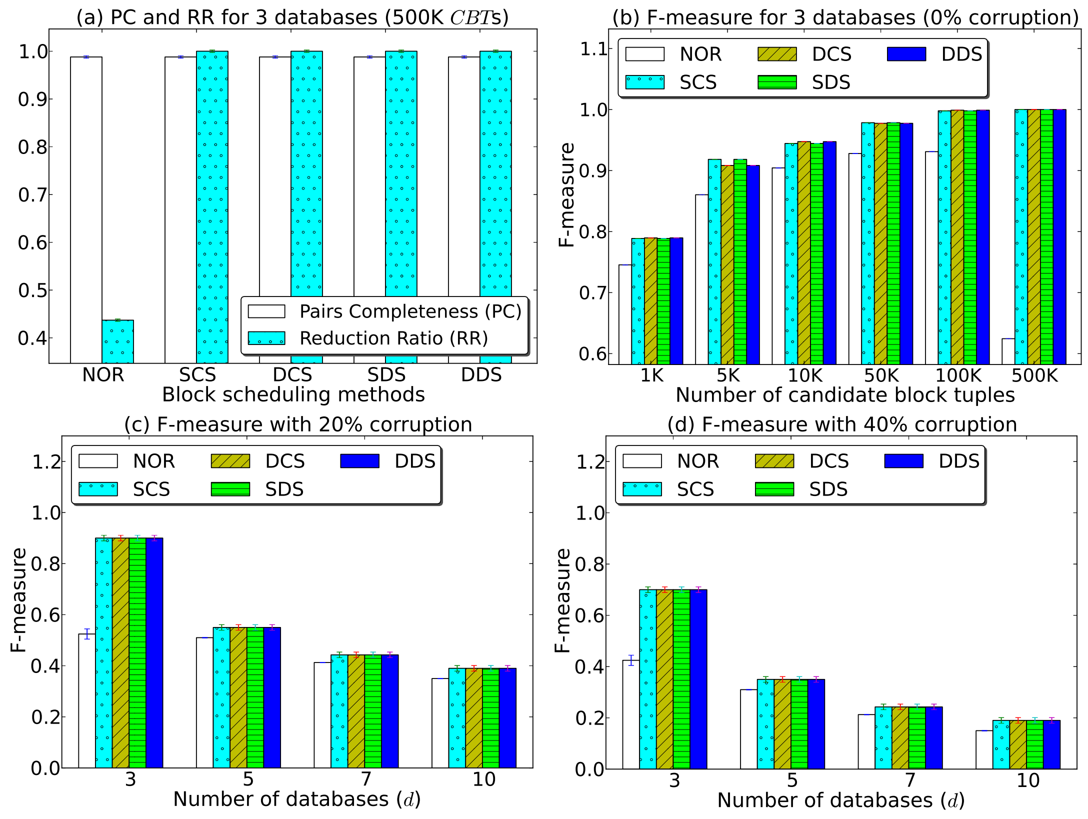


Figure 9.6: (a) Pairs completeness (PC) and reduction ratio (RR), and (b) F-measure (FM) with different scheduling methods for three databases, and FM with different number of database for different scheduling methods with (c) 20% and (d) 40% corrupted databases. Note that plots have different y-axis scales.

Figures 9.6 (c) and (d) show the effectiveness of our approach in terms of F-measure (FM) with different number of databases with 500,000 *CBT*s. These figures show that the scheduling approaches can achieve better FM compared to NOR because scheduling improves the RR considerably by removing redundant record pair comparisons. Therefore, the use of our proposed meta-blocking approach before the comparison and classification step in a linkage application can achieve high efficiency and scalability with no loss in effectiveness.

9.5.3 Privacy

To evaluate the privacy of our approach we conducted the cryptanalysis attack described in Section 5.5 with each scheduling technique for each generated block of a database with 1 million records. We used the same parameter settings described in Section 5.5 to conduct this cryptanalysis attack. As shown in Figure 9.7 (a), most of the re-identification guesses resulted in wrong and no guesses. This indicates that an attacker could not re-identify an attribute value in a given block because it does not contain enough frequency information to identify q-grams that are encoded in BFs.

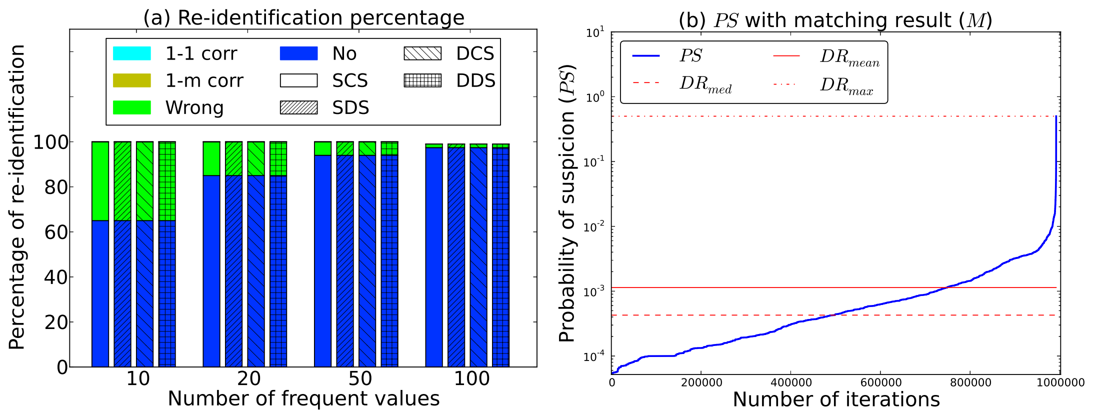


Figure 9.7: (a) The cryptanalysis attack performed upon the generated blocks and (b) probability of suspicion and disclosure risk values with block pair comparisons (matching results) for a database with 1,000,000 records. Note that plots have different y-axis scales.

As shown in Figure 9.7 (b), we computed average probability of suspicion (PS) values (for the frequency linkage attack described in Section 2.5 on page 38) for the comparisons of block pairs of three databases with 1 million records. The PS values are measured for the matching records that are resulted in each block pair comparison in a given scheduling iteration. As can be seen in this figure, the PS value of a record increases in the later scheduling iterations of any given scheduling method. This is because the later block pair comparisons result in very small block sizes (set of matching records) which are added as new blocks into the blocking graph **G**. This could potentially increase the risk of frequency attacks in the centralised and distributed comparison scenarios. To overcome the risk of such attacks, a secure comparison and classification technique needs to be used in our scheduling techniques. However, it is important to note that such a privacy breach can only happen at the comparison and classification step which is outside of our meta-blocking approach.

9.6 Summary

In this chapter we have proposed a novel scalable meta-blocking approach for multidatabase privacy-preserving record linkage (MD-PPRL). Our approach accepts a list of candidate block tuples (that consist of blocks from multiple databases) as input and orders pairs of blocks in these tuples to minimise the number of repeated and superfluous record pair comparisons, where the latter are comparisons of records with record pairs that have previously been classified as non-matches. Our approach uses a graph structure to schedule the comparison of pairs of blocks based on a ranking of edge weights. As explained in Section 9.3, we propose five different weighting schemes that can be used to calculate the weights of pairs of blocks, and four scheduling methods to order pairs of blocks for comparisons.

As discussed in Section 9.5 we evaluated our approach with large databases which indicated our approach is scalable with both the size and the number of databases. Our experimental results showed that the proposed approach can significantly reduce the number of record pair comparisons required which improves the efficiency of the overall linkage.

Chapter 10

Comparative Evaluation of Proposed Techniques

As we have described in Chapter 1, multidatabase privacy-preserving record linkage (MD-PPRL) inherits several challenges. In Chapter 4 we have introduced a blocking framework for MD-PPRL to address the challenges related to blocking. We have proposed several novel techniques that can be incorporated with our proposed blocking framework in Chapters 5 to 9. In this chapter, we comparatively evaluate our proposed blocking approaches with several state-of-the-art techniques that can be used for blocking in a MD-PPRL context. We first introduce the state-of-the-art solutions used in this comparative evaluation and their parameter settings in Section 10.1. Next, we present the empirical evaluation results and discuss our findings in Section 10.2. Finally, we summarise our evaluation in Section 10.3.

10.1 Introduction

In MD-PPRL, blocking plays a major role in the linkage process. Blocking reduces the exponential and quadratic candidate record comparison space that occurs in the linkage process when the number and size of the databases to be linked are increasing, respectively, by removing those record tuples that are unlikely to be matching. As we have discussed in Section 1.3, blocking in MD-PPRL can be performed with or without a *linkage unit* (LU). The blocking techniques proposed in Chapters 5 (labelled as SBT for *single-bit tree* based blocking) and 6 (labelled as SCC and HCC for *standard* and *hierarchical canopy clustering* based blocking, respectively) can be used in a MD-PPRL context without a LU, while in Chapter 7 we proposed a distributed blocking scheme (labelled as DBS for *distributed blocking scheme*) that uses a LU to identify the blocks that need to be compared. In Chapter 8 we proposed a blocking technique (labelled as SGB for *subgroup blocking*) that identifies candidate block tuples (*CBTs*) for different subgroup combinations across a set of databases that need to be linked. The meta-blocking technique (labelled as MBS for *meta-blocking scheme*) proposed in Chapter 9 that schedules block pairs for comparisons can be used in MD-PPRL for removing redundant record pair comparisons. We use the same parameter settings described in these chapters for the empirical evaluation describe in this chapter.

Table 10.1: Notation and terminology used in this chapter

$\mathbf{CBT}$	List of candidate block tuples
$\mathbf{D}$	A database
$\mathbf{G}$	Candidate blocking graph
CBT	Candidate block tuple
d	Number of databases
F	List of fixed databases
$\mathbb{G}$	Publicly available (global) database
SG	A subgroup
R	A record
ACG	Apriori based candidate generation algorithm described in Algorithm 10.1
BF	Bloom filter
BK	Blocking key
BKV	Blocking key value
DBS	Distributed blocking scheme described in Chapter 7
DCG	Depth first based candidate generation algorithm described in Algorithm 8.2
DO	Database owner
FPS	Frequent pair blocking scheme technique proposed in [119]
HBH	Hamming based hashing technique proposed in [56]
HCC	Hierarchical canopy clustering based blocking approach described in Chapter 6
LSH	Locality sensitive hashing
LU	Linkage unit
MBS	Meta-blocking approach described in Chapter 9
MD-PPRL	Multidatabase privacy-preserving record linkage
NOR	Normal block scheduling method
PHO	Phonetic encoding based blocking used in [36, 71]
RAD	Random block scheduling method
SBT	Tree based blocking appproach described in Chapter 5
SCC	Standard canopy clustering based blocking approach described in Chapter 6
SGB	Subgroup blocking approach described in Chapter 8

10.1.1 Comparative Evaluation Techniques

In the following, we compare and evaluate our proposed solutions with some of the state-of-the-art solutions with regard to efficiency (scalability), effectiveness (blocking quality), and privacy. We prototyped all the comparison techniques in Python version 2.7.3 (as detailed in Section 4.4.3 on page 84), and all the experiments were conducted on the datasets (see Table 4.3 on page 79) described in Section 4.4.1. We use the *given name*, *surname*, *city*, and *postcode* attributes as blocking key (BK) attributes. Table 10.1 summarises the notations we use in this chapter. We next describe the techniques used in this comparative evaluation in more details.

- **Block Generation**

  For comparative evaluation we use a traditional and two state-of-the-art blocking solutions [56, 71, 119]. The traditional approach we use for comparative evaluation is a standard blocking approach [36, 71] using phonetic encoding [35, 110] (labelled as PHO). In this approach we use Soundex [35] as encoding function for the given name, surname and city attributes, while for the postcode attribute the first three digits of a postcode value are used as BKs.

We assume a LU is employed in this approach to conduct blocking. All the database owners (DOs) send the phonetic encodings of the attribute values of each record of their databases to the LU along with the record identifiers. Then, the LU groups the records from different databases together based on the encoded values of one or a combination of record attributes. For example, if the given name and surname attributes are used as BKs, each generated block will contain only records that have the same concatenated Soundex encodings of given name and surname attribute values.

To illustrate how PHO works, let us assume three given records R_1, R_2, and R_3 with respective values *John Smith*, *Jone Smith*, and *Peter Smith* for given name and surname attributes. If we use the concatenated Soundex encodings of given name and surname attribute values as blocking key values (BKVs), the records R_1 and R_2 will be assigned to the same block because both R_1 and R_2 would result in *J500S530* as their corresponding BKVs. However, the record R_3 will be assigned to a different block as its BKV would result in *P360S530*. Such phonetic encoding based blocking technique is illustrated in Figure 8.2 on page 159.

As a second baseline method, we use an adapted version of a state-of-the-art Hamming distance based locality sensitive hashing (LSH) private blocking technique with a LU proposed by Durham [56] (labelled as HBH for *Hamming based hashing*). As reviewed in Section 3.1 on page 41, in this approach all DOs first encode the records in their databases into Bloom filters (BFs) which are then sent to a LU to group into blocks using LSH.

As the third comparative technique, we use the state-of-the-art frequent pair blocking scheme recently proposed by Karapiperis and Verykios [119] (labelled as FPS for *frequent pair scheme*). As detailed in Section 3.1, this approach also uses a Hamming distance based LSH technique to assign records into independent blocking groups. We use the parameter settings for the HBH and FPS methods in similar ranges as used by the authors of these private blocking techniques. Please note that apart from these three blocking techniques the recent multi-party PPRL blocking approach proposed by Han et al. [91] can only be applied to numerical attributes, therefore, can not be compared directly with our blocking approaches.

In the HBH method the parameters set as, the number of iterations (μ) = 40, the number of hash functions (n_h) = 30, the length of BFs (l_{bf}) = 1,000 bits, and the number of bits to be sampled from the BFs at each iteration (σ) = 45. In the FPS technique, n_h and l_{bf} are set to 30 and 1,000 bits, respectively, and the Hamming distance (ϑ), the confidence parameter (δ), and number of blocking groups (L_C) are set to 100 bits, 0.01, and 20, respectively.

- **Subgroup Blocking**

  For the comparative evaluation of our subgroup blocking approach (in Chapter 8), we propose an Apriori based candidate block tuple generation algorithm (labelled as ACG) because to best of our knowledge there are no other subgroup

Algorithm 10.1: Apriori based Candidate Generation

```
Input  : G  - Undirected candidate graph (as described in Section 8.3)
         $w_t$ - Weight threshold
         $g_\alpha$ - Minimum subgroup size
         $g_\beta$ - Maximum subgroup size
         $d$  - Number of databases
         $F$  - Set of fixed databases
Output: CBT - Inverted index of subgroup candidate block tuples
CBT $\leftarrow \{\}$, $sg \leftarrow 2$, SG $\leftarrow \{\}$                      // Initialise variables
if $sg < g_\alpha$ then                     // Check if minimum subgroup size is larger than $sg$
    SG.add(genAdditionalgroupComb($sg, g_\alpha, d, F$))       // Generate auxiliary combinations
end
SG.add(genSubgroupComb($g_\alpha, g_\beta, d, F$))          // Generate required group combinations
foreach $SG \in$ SG[2] do                  // Process every subgroup combination of size 2
    $C_2$.add(getEdges(G, $SG, w_t$))                              // Process the edges in G
end
CBT[2] $\leftarrow C_2$                           // Add the generated cliques of size 2 to CBT
$sg \leftarrow 3$                                                      // Increment $sg$ to 3
while $sg \leq g_\beta$ and $C_{sg-1} \neq \emptyset$ do            // Iterate over every subgroup combination
    $C_{sg} \leftarrow$ genCliques(G, SG[$sg$], $w_t, C_{sg-1}$)             // Compute cliques of size $sg$
    if $g_\alpha \leq sg$ then
        CBT[$sg$] $\leftarrow C_{sg}$                   // Add the generated cliques of size $sg$ to CBT
    end
    $sg \leftarrow sg + 1$                                        // Increment subgroup size by 1
end
return G $= (V, E)$
```

blocking techniques available for MD-PPRL. As outlined in Algorithm 10.1, we adapted Algorithm 8.2 on page 164 to incorporate the Apriori technique [1, 102] to generate *CBT*s. This approach uses a breadth first search over the candidate graph **G** (see Definition 8.1 on page 162) in the *CBT* generation. In this approach the *CBT* generation starts with a minimum subgroup size of 2 and iteratively extends the *CBT* generation for larger subgroup sizes by incrementing the subgroup size by 1 until it reaches the maximum subgroup size g_β.

In situations where a user specifies $2 < g_\alpha$ our approach still requires to generate sets of subgroups combinations of sizes 2 to $g_\alpha - 1$ (line 3). For example to generate *CBT*s for the subgroup combination $(\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C)$, Algorithm 10.1 needs to generate *CBT*s for either $(\mathbf{D}_A, \mathbf{D}_B)$, $(\mathbf{D}_A, \mathbf{D}_C)$, or $(\mathbf{D}_B, \mathbf{D}_C)$. These additional subgroups are required due to the Apriori based clique (*CBT*) generation process.

Algorithm 10.1 follows an iterative approach to identify cliques of size sg from the cliques of size $sg - 1$ that are identified in the previous iteration (starting from pairs, i.e. $sg = 2$). The function *genAdditionalgroupComb()* computes the additional subgroup combinations required based on g_α, d, and F (line 3).

Similar to Algorithm 8.2, the function *getEdges()* (in lines 6 to 8) of Algorithm 10.1 identifies the trivial cliques for each subgroup SG of size $sg = 2$ which are the set of edges $e \in E$ of $\mathbf{G}$ that satisfy the weight threshold w_t. In lines 9, these computed edges are added to the inverted index **CBT** using each corresponding subgroup combination as a key. In lines 11 to 13, the function *genCliques()* uses a breadth-first search over $\mathbf{G}$ to identify the cliques of size sg. These are then added to the set C_{sg}.

In the next iteration $(sg + 1)$, these identified cliques C_{sg} are used to generate the set of CBTs for the next sets of subgroups by increasing the subgroup size sg by 1 (line 16). This clique generation process in Algorithm 10.1 continues until sg reaches g_β and $C_{sg} \neq \emptyset$ (line 11). All the generated CBTs are added to an inverted index **CBT** appropriately for under each subgroup combination (line 14). We use the same parameter setting described in Section 8.5 for the evaluation of Algorithm 10.1.

- **Meta-blocking**

  For the comparative evaluation purposes of our meta-blocking approach we use two naïve block scheduling approaches, normal record set comparison (labelled as NOR) and random record set comparison (labelled as RAD). The first is the standard approach of comparing each record in a block with every record in all other blocks of a CBT (as we have described in Section 9.5), while the second performs record pair comparisons between randomly ordered block pairs of a given collection of CBTs. Note that there are no other block scheduling and meta-blocking techniques for MD-PPRL which can be directly compared with our meta-blocking approach.

10.2 Experimental Evaluation

In this section we present the empirical evaluation results and discuss our findings with regard to the three main properties of scalability, blocking quality, and privacy.

10.2.1 Block Generation

Figure 10.1 shows the scalability of the seven private blocking approaches in terms of runtime. We measure the average runtime required per database and the total runtime required for different number of databases. As can be seen in this figure, PHO and HLSH require less runtime than the other blocking approaches. SBT, SCC, and HCC require more runtime because they require frequent communication between DOs while DBS runs faster due its independent nature in the block generation process. However, as can be seen all of our blocking approaches are scalable to large and increasing number of databases.

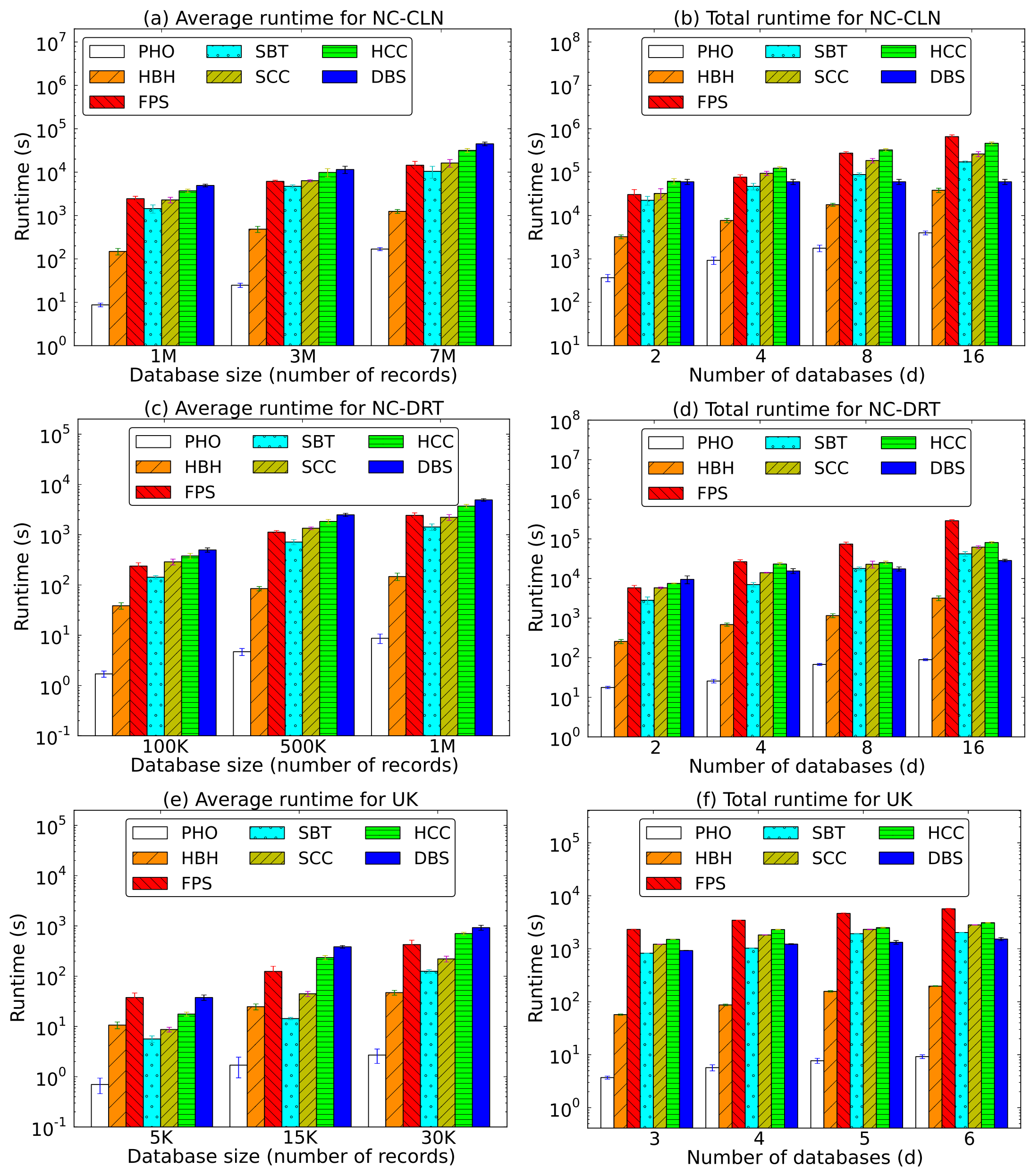


Figure 10.1: Average and total runtime results for the seven blocking approaches, where plots (a) and (b) are for the NC-CLN (clean), (c) and (d) for the NC-DRT (dirty), and (e) and (f) for the UK datasets. (a), (c), and (e) show the average runtime (left column) for block generation with different sizes of databases. (b), (d), and (f) show the total runtime (right column) requires with different number of databases for different blocking approaches. K and M represent 1,000 and 1,000,000 records, respectively. Note that plots have different y-axis scales.

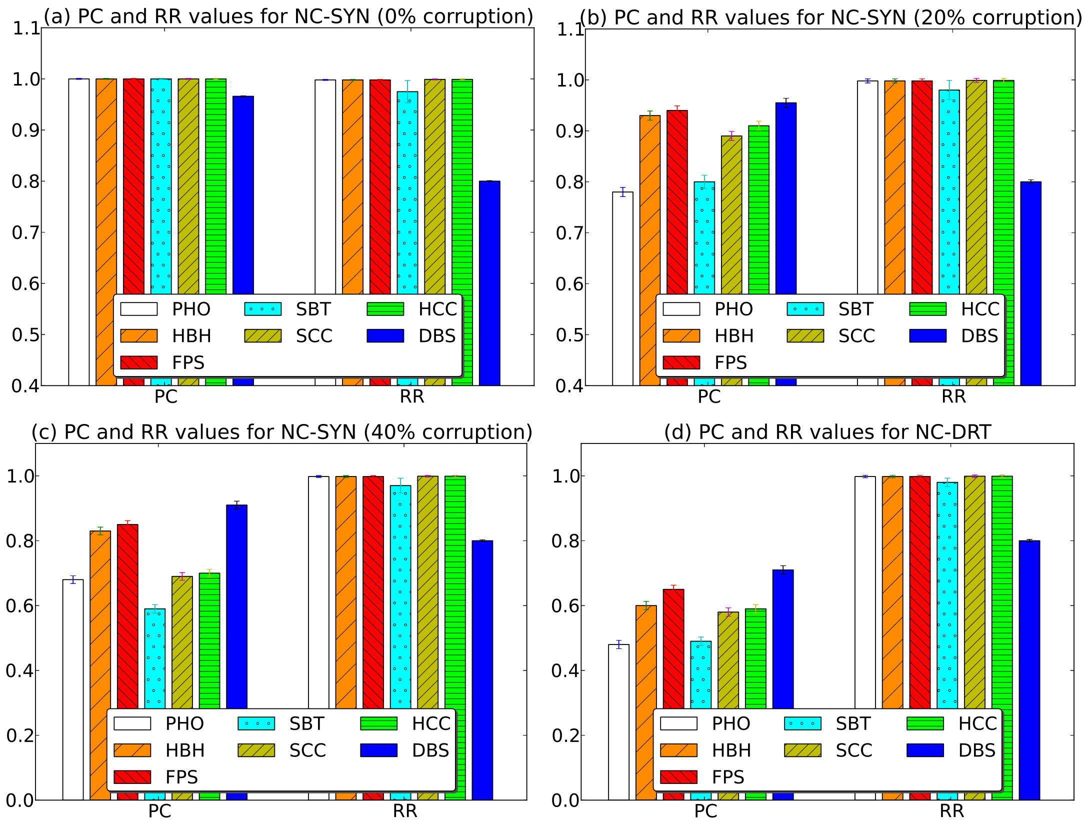


Figure 10.2: Average pair completeness (PC) and reduction ratio (RR) with different blocking techniques for the NC-SYN datasets with (a) 0%, (b) 20%, and (c) 40% corruption levels and with (d) the NC-DRT dataset. Note that plots have different y-axis scales.

The efficiency and effectiveness of blocking, measured by reduction ratio (RR) and average pairs completeness (PC), respectively, of the seven blocking approaches are compared on the NC-SYN and NC-DRT datasets in Figure 10.2. As can be seen in Figure 10.2 (a), all these blocking approaches achieve higher PC and RR rate for 0% corrupted NC-SYN dataset. Figures 10.2 (b), (c), and (d) show the trade-off between the scalability (efficiency) and quality (effectiveness) of blocking in the presence of dirty data. As can be seen in these figures, the DBS approach achieves the highest PC at the cost of some reduction in RR, while the other approaches comparatively have lower PC with RR being almost 1.0.

Finally, we evaluated the privacy protection of these blocking approaches using the evaluation measures presented in Section 4.4 on page 81 and conducted the crypt-analysis attack described in Section 4.5 on the NC-ORG dataset. We used the original dataset ($\mathbf{D}$) as the global dataset ($\mathbb{G}$), i.e. $\mathbb{G} \equiv \mathbf{D}$, for privacy evaluation under the worst case assumption and minimum block size (b_{min}) is set to 1,000.

Figure 10.3 (a) illustrates the comparison of the size of blocks generated by the seven private blocking approaches shown in a box-and-whisker plot. As can be seen

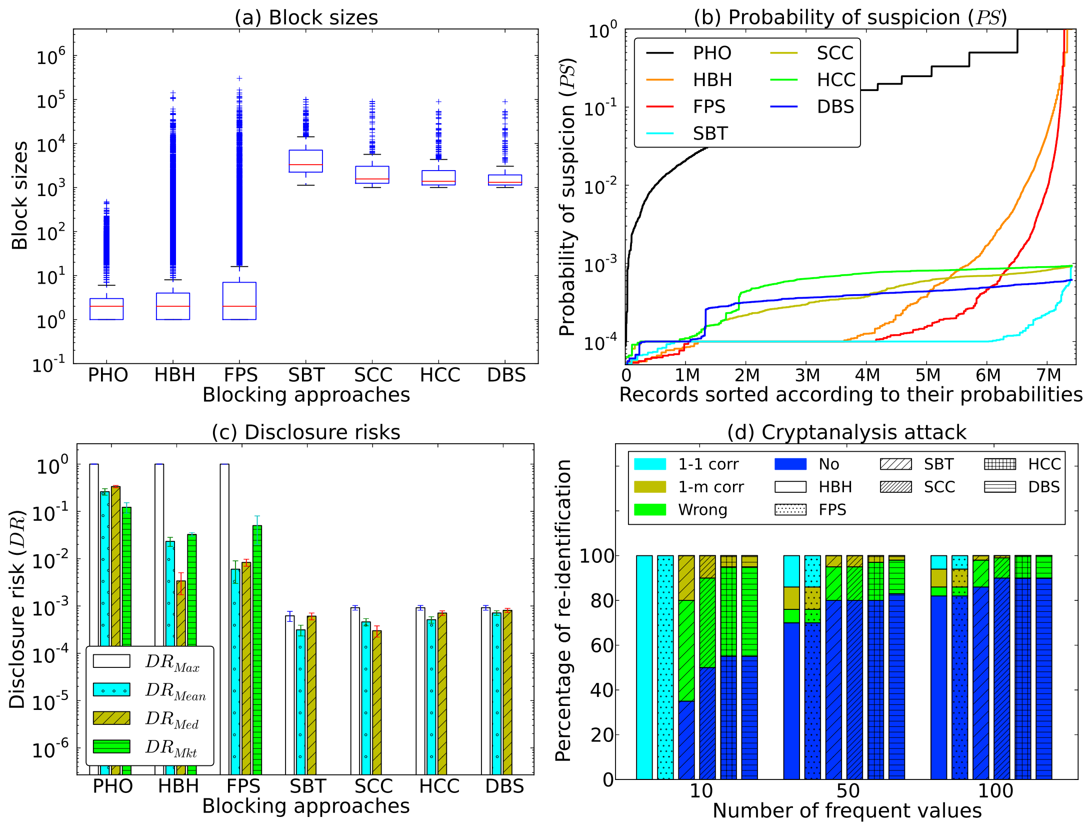


Figure 10.3: (a) Block sizes, (b) probability of suspicion (PS), (c) disclosure risk (DR) values, and (d) re-identification percentages of cryptanalysis attack with the NC-ORG dataset. Note that plots have different y-axis scales and M represents 1,000,000 records.

in this figure, the PHO, HBH, and FPS approaches generate that blocks contain only 1 record compared to our proposed blocking approaches SBT, SCC, HCC, and DBS. All these blocks might reveal private information and thus these approaches should not be used for blocking in MD-PPRL. We noted that HBH and FPS generate overlapping blocks of smaller sizes and the variance between block sizes is comparatively very high. Our proposed blocking approaches have lower variances between the block sizes which makes a frequency attack using block sizes more difficult, where the DBS approach shows the lowest variance compares to the other approaches. This indicates that independent blocking provides the DOs more control over their block generation process that improves the privacy against frequency attacks.

It is important to note that in SBT, SCC and HCC only the DOs are participating in the block generation process. In DBS the LU only receives a set of block representatives of the generated blocks by each DO with regard to their databases. Hence, in DBS a frequency attack by an external attacker is not possible. This is because the attacker does not know the parameter values that can be used to mount a frequency attack using $\mathbb{G}$ and therefore learning the BKs or their values in $\mathbf{D}$ is difficult [212].

Figure 10.3 (b) shows the distributions of probability of suspicion (PS) values in the NC-ORG dataset blocked by the seven private blocking approaches. The records in the datasets are sorted according to their PS values. As can be seen in this figure, DBS generates the lowest probability of suspicion curve compared to other blocking approaches. However, the maximum PS of PHO, HBH, and FPS goes higher (PS = 1) because many of their generated blocks contains only one record.

Figure 10.3 (c) shows a comparison of disclosure risk (DR) measures (DR_{Max}, DR_{Mean}, DR_{Med}, and DR_{Mkt} calculated from the probability of suspicion values PS, as shown in Figure 10.3 (b) and explained in Section 4.4 (on page 81) of the seven private blocking approaches on the NC-ORG dataset. As shown in this figure our proposed blocking approaches have much lower values for DR measures compared to PHO, HBH, and FPS. We noted that DR_{Max} of PHO, HBH, and FPS is 1.0 even when we use the all four attributes (the *given name*, *surname*, *city*, and *postcode*) as BKs. We also noted that DR_{Mkt} of PHO, HBH, and FPS reaches 0.122, 0.034, and 0.051, respectively, as more blocks are generated by these approaches contain only one record, while our blocking approaches has DR_{Mkt} as 0.

In the cryptanalysis attack, we assumed the LU would act as an attacker and tries to re-identify an attribute value encoded in the Bloom filters (BFs). We conducted the attack upon encoded BFs of the six blocking approaches (except for PHO as it uses Soundex as the encoding mechanism) that only contain values of *given name* under the worst case assumption. This is because the re-identification of attribute values becomes harder as BFs are encoded with more attribute values [39]. We conducted the attack for 10, 50, and 100 most frequent attribute values.

As shown in Figure 10.3 (d), an attacker (the LU) can correctly re-identify all values of *given name* when the number of frequent values is 10 while it can still re-identify 6% of 100 frequent names of the blocks generated by HBH and FPS. This is because each DO sends its encoded database (BFs) to the LU to generate blocks. As shown in this figure, most of the re-identification guesses of our blocking approaches resulted in wrong or no guesses. Also, such an attack could not really happen in our blocking approaches as the DOs do not reveal their encoded BFs to any other party. The re-identification results shown in this figure indicate that conducting such a cryptanalysis attack to correctly re-identify attribute values at the block level of our blocking approaches is impossible.

Overall this privacy evaluation shows that our blocking approaches provide better privacy compared to these other baseline blocking approaches, making our blocking approaches more practical and secure to be used in real MD-PPRL applications.

10.2.2 Subgroup Blocking

To evaluate our subgroup blocking (SGB) approach we use the sets of blocks generated for each DO using DBS. We follow the same set of steps described in Section 7.3 on page 136 to generate Min-Hash signatures as block representatives.

Figure 10.4 illustrates the scalability of SGB in terms of runtime for different subgroup blocking scenarios described in Section 8.3 on page 159. We measure the

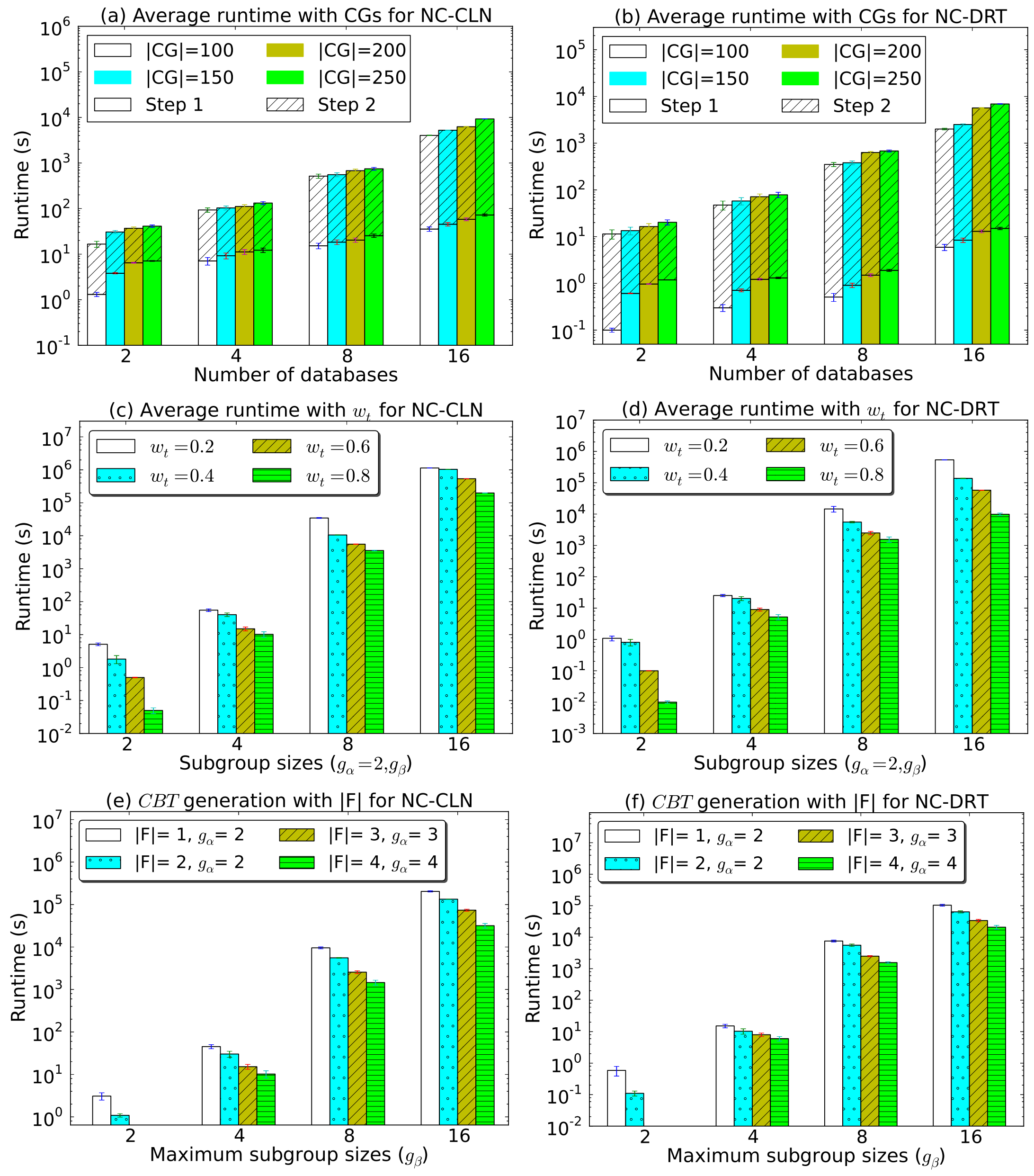


Figure 10.4: Average runtime results for subgroup blocking approach (SGB), where plots (a), (c), and (e) show the results for the NC-CLN datasets, and plots (b), (d), and (f) show the results for the NC-DRT datasets. (a) and (b) show the average runtime for candidate grouping (step 1) and candidate graph construction (step 2) of SGB with different number of candidate groups (*CG*s). (c) and (d) show the average runtime with different weight thresholds (w_t) for different subgroup sizes, and (e) and (f) the average runtime required for the depth-first candidate generation (DCG) algorithm with different number of databases fixed ($|F|$) in subgroup combinations. Note that plots have different y-axis scales.

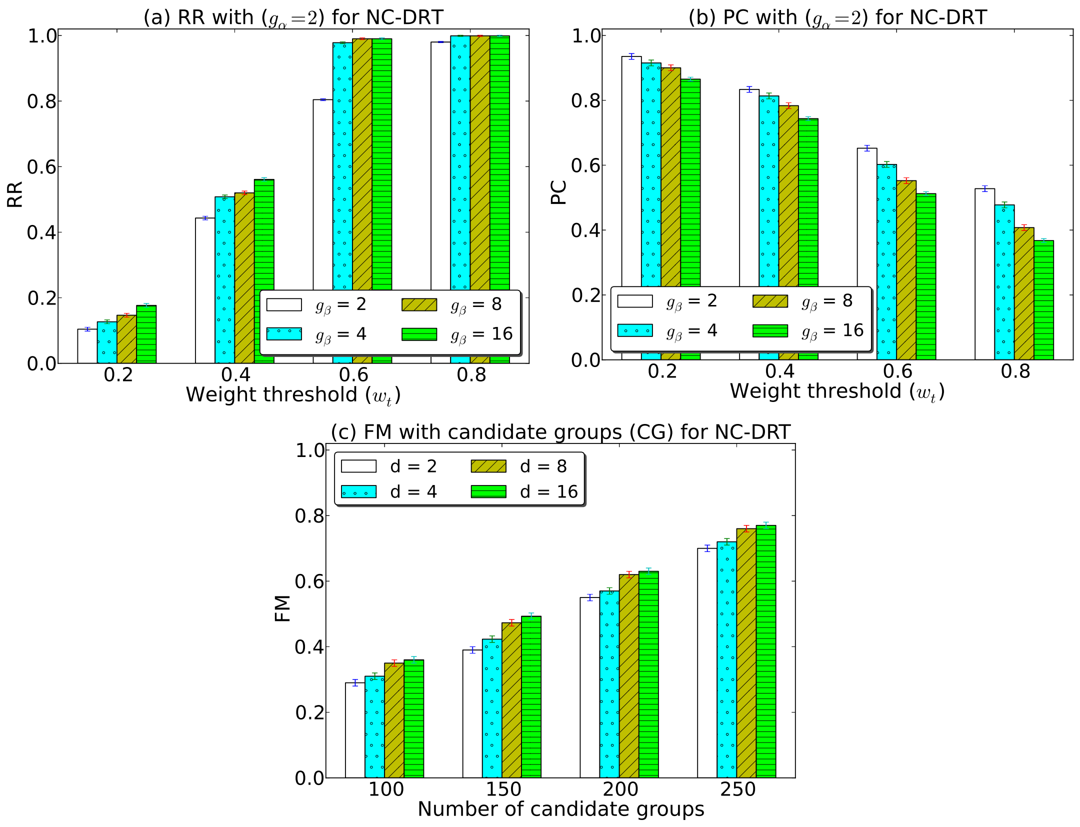


Figure 10.5: (a) Reduction ratio (RR) and (b) pairs completeness (PC) for different weight thresholds (w_t) for different number of subgroup combinations, and (c) combined F-measure (FM) with different number of candidate groups (CGs) with different number of databases (d) for the NC-DRT datasets.

runtime required for each step in SGB for different d and different minimum (g_α) and maximum (g_β) subgroup sizes for NC-CLN and NC-DRT datasets.

As shown in Figure 10.4 (a) and (b), the average runtime required for steps 1 and 2 of SGB increases linearly with the number of databases d. We observed that the average runtime also increases linearly with the number of candidate groups ($|CG|$), which suggests that more block pairs are being generated across these databases which increases the number of edges in the candidate graph $\mathbf{G}$.

As expected the average runtime decreases with an increase in the weight threshold (w_t) as shown in Figures 10.4 (c) and (d). As the weight constraint w_t increases, edges with lower similarity between their corresponding blocks are not considered in the CBT generation. However, the runtime increases linearly with the size of subgroups as more combinations are considered in the CBT generation process. Figures 10.4 (e) and (f) show the average runtime required for generating CBTs for different subgroup sizes with a certain set of databases (F) fixed. As expected the runtime decreases when more databases are included in F since the number of group combinations considered in each subgroup size decreases with the size of F.

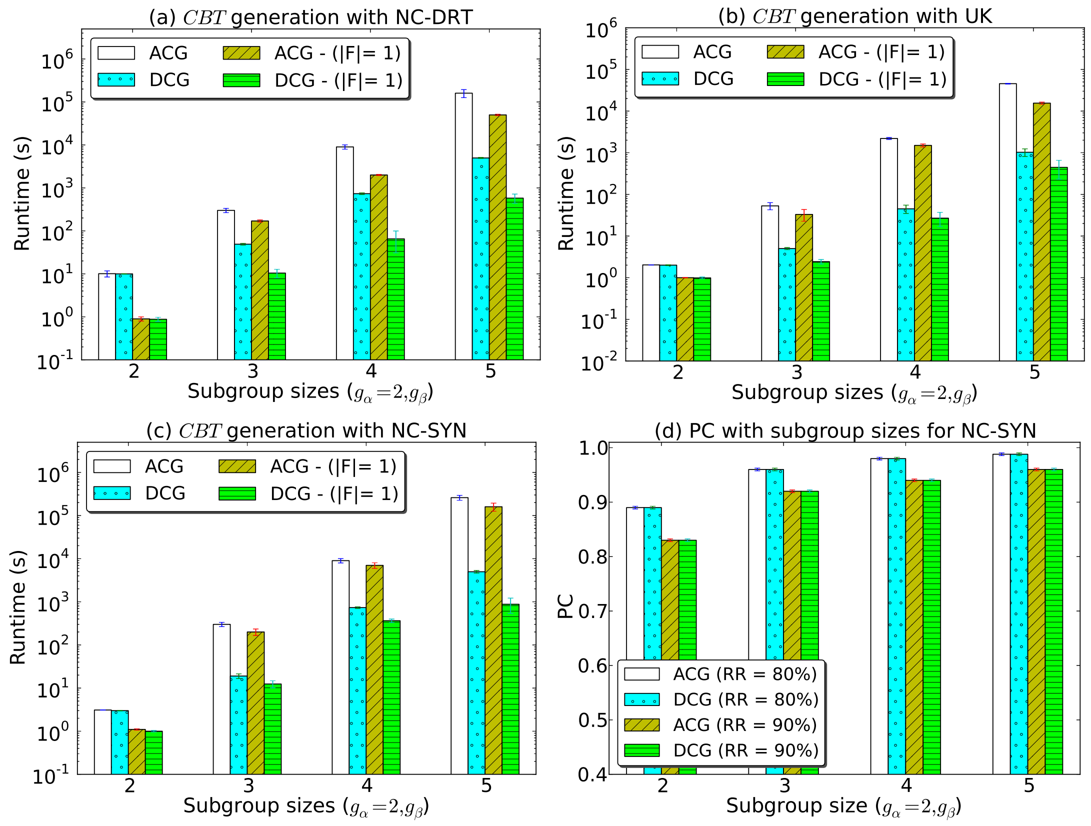


Figure 10.6: Average runtime for candidate block tuple (CBT) generation in our subgroup blocking approach for different subgroup combinations with (a) the NC-DRT and (b) the UK datasets, (c) average runtime for CBT generation, and (d) pairs completeness (PC) with the Apriori (ACG) and depth-first (DCG) candidate generation algorithms for the NC-SYN datasets.

Figure 10.5 (a) illustrates the scalability of our subgroup blocking approach in terms of reduction ratio (RR). As can be seen in this figure, RR increases with w_t which suggests that less CBTs are generated for a given subgroup size. As can be seen in Figure 10.5 (b), an increase in w_t results in pairs completeness (PC) to decrease as true matching record pair comparisons are missed due to the decreased number of block tuple comparisons.

Figure 10.5 (c) illustrates the blocking quality of SBG in terms of combined F-measure (FM) for different candidate groups (CG) with different number of databases (d). As shown in this figure, FM increases with $|CG|$ which suggests that CBT generation becomes more fine grained as **G** becomes more dense. However, an increment of $|CG|$ could potentially increase the runtime of SGB as shown in Figure 10.4.

In step 3 of our SGB approach, the generation of CBTs can be conducted by using the Apriori based candidate generation (ACG) algorithm (Algorithm 10.1 discussed in Section 10.1). In terms of time and space complexity ACG would require a complexity of $O(n^{g_\alpha})$ [102] if each of the g_α databases generates n blocks. In line 12 of Algorithm 10.1, the generation of cliques of size sg depends on the number of $(sg-1)$ size cliques (C_{sg-1}) generated previously. Hence, the complexity of ACG is $O(E + \sum_{g_\alpha=3}^{g_\beta} \binom{g_\alpha}{g_\beta} \cdot |C_{g_\alpha-1}|^2)$, where $|E|$ is the number of edges in $\mathbf{G}$. This becomes computationally infeasible when n, g_α, and g_β are increasing.

As shown in Figure 10.6 we compare the runtime required for the CBT generation for different subgroup sizes with the ACG and DCG based candidate generation algorithms for datasets NC-DRT, UK, and NC-SYN. As can be seen in this figure, DCG requires less runtime compared to ACG, which validates that DCG is more efficient in CBT generation. We also measured the memory required for the CBT generation where in average our DCG algorithm only uses below 10% of the total memory required by ACG. We were unable to conduct experiments for ACG with subgroup size larger than 5 due to its memory requirements. Such memory consumption occurs due to the Apriori nature in CBT generation because it requires the previously generated list of block tuples for the subsequent candidate generation iterations.

As shown in Figure 10.6 (d), we also noted that both ACG and DCG achieve the same PC values which suggests that both ACG and DCG generate the same set of CBTs for a given RR value. As can be seen in Figure 10.6, AGC is still competitive with our proposed iterative deepening depth-first based technique if the number of blocks from each database remains small. However, with the increase of subgroup sizes DCG outperform ACG by generating CBTs more efficiently which suggests that in practice DCG is more appropriate in MD-PPRL applications than ACG.

10.2.3 Meta-Blocking

To evaluate our meta-blocking (MBS) approach with baseline techniques we use the lists of CBTs of subgroup combinations 2, 4, 8, and 16 for the NC-CLN and NC-DRT datasets, and 3, 4, 5, and 6 for the UK dataset generated with a weight threshold w_t set to 0.5. We name the four different block comparison scheduling methods as SCS for *static centralised scheduling*, DCS for *dynamic centralised scheduling*, SDS for *static distributed scheduling*, and DDS for *dynamic distributed scheduling*.

Figure 10.7 illustrates the scalability of MBS in terms of the number of record comparisons required for different block scheduling methods with different number of databases and weighting schemes. As can be seen from Figures 10.7 (a), (c), and (e) the total number of record comparisons required by each scheduling method scales linearly with the number of databases. However, the number of record comparisons increases with the number of databases as more block pairs need to be processed.

Figures 10.7 (b), (d), and (f) illustrate the average number of record pair comparisons performed per database with different weighting schemes and scheduling methods (see Section 9.3 on page 179). Similar as in Section 9.5, DDS leads to a higher reduction in record pair comparisons compared to the other scheduling methods as DDS compares block pairs completely distributed leading to an improvement in the overall efficiency of MBS. Similarly, the comparison (w_{comp}) based weighting scheme performs better as it is capable of removing more record pair comparisons compared to the cardinality (w_{card}, matching (w_{match}), and similarity (w_{sim}) based weighting

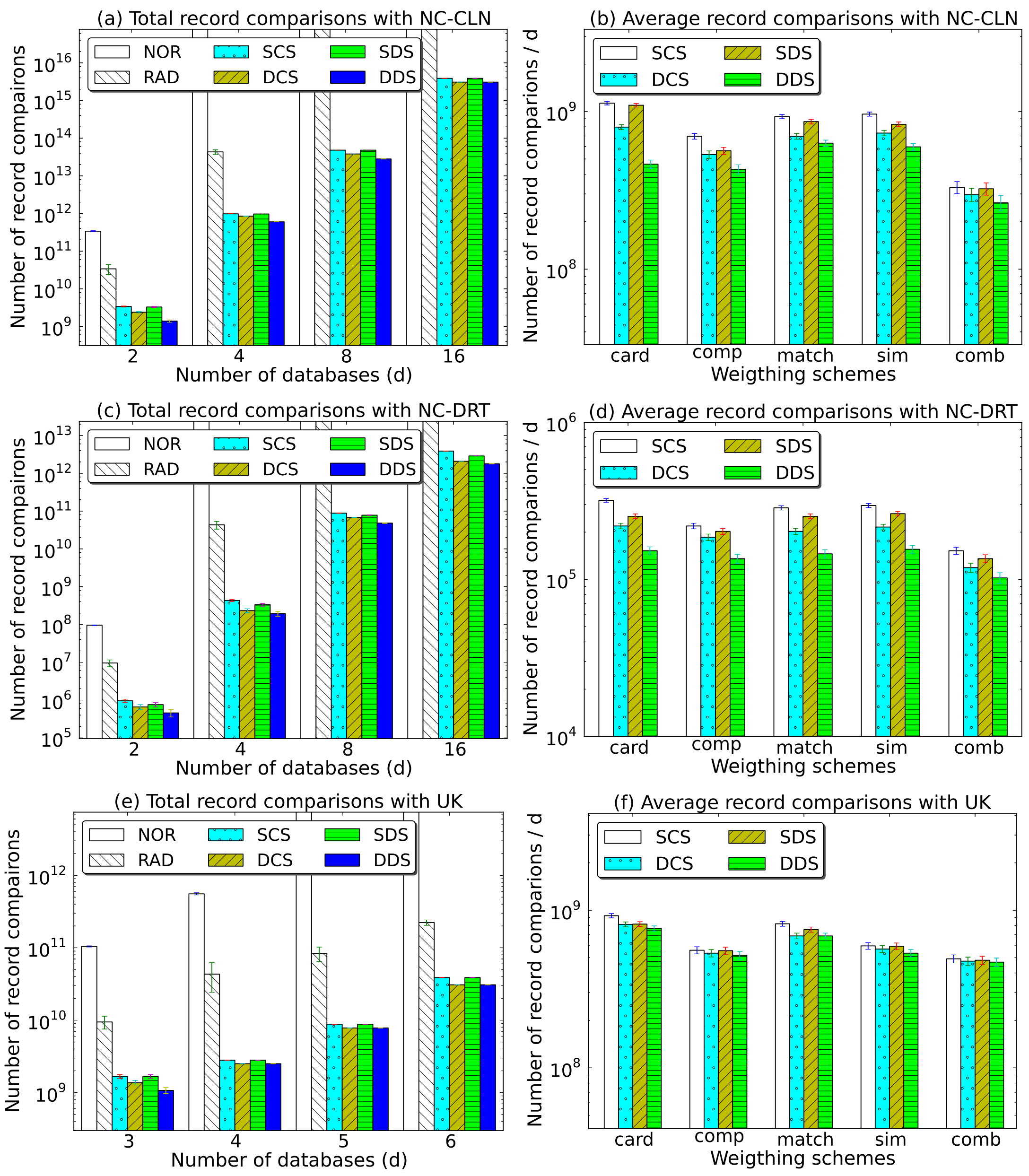
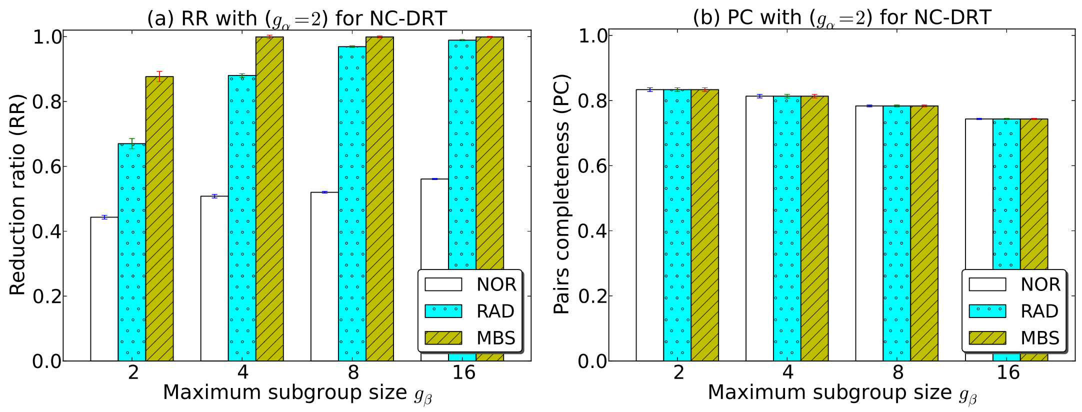


Figure 10.7: Total and average number of record comparisons with our meta-blocking approach with different scheduling techniques. Results show (a) and (b) for the NC-CLN, (c) and (d) for the NC-DRT, and (e) and (f) for the UK datasets. Note that plots have different y-axis scales.

schemes. This is because w_{comp} selects the block pairs with the smallest number of record pair comparisons first and therefore removes more non matching record pairs at an early stage of the comparison process. These figures also show that a better reduction can be achieved when weighting schemes are combined.



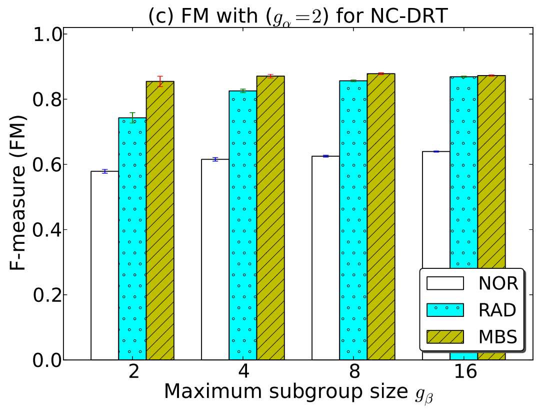


Figure 10.8: (a) Reduction ratio (RR), (b) pairs completeness (PC), and (c) combine F-measure (FM) for different number of subgroup combinations for the NC-DRT datasets.

Figure 10.8 illustrates the effectiveness of our MBS approach in terms of RR, PC, and FM with the NC-DRT dataset. As can be seen in this figure, MBS increases the RR considerably compared to the NOR and RAD baselines. Figure 10.8 (a) shows our scheduling techniques reduce the number of record pair comparisons included in the sets of CBTs, however, they do not reduce the number of true matches in the compared pairs of blocks as shown in Figure 10.8 (b). Figure 10.8 (c) shows how the use of scheduling can improve the effectiveness of MD-PPRL even for large numbers of databases compared to NOR and RAD which lead to lower FM values due to their large number of record pair comparisons. Overall, based on the discussed results, we conclude that the scheduling methods perform more effectively than the NOR and RAD approaches without sacrificing the effectiveness of the linkage process.

10.3 Summary

In this chapter, we have conducted a comprehensive evaluation and comparison of our proposed MD-PPRL approaches with several existing state-of-the-art techniques on large real-world databases using the experimental setup described in Chapter 4.

The experimental results on these databases showed that our proposed blocking approaches achieve better blocking quality and scalability while providing strong privacy against frequency and cryptanalysis attacks compared to several existing solutions. Our subgroup blocking approach is scalable with the size of subgroups and it outperforms a baseline approach in terms of subgroup candidate blocks generation, while our meta-blocking approach improves the efficiency of the overall MD-PPRL process without any loss in effectiveness.

Further work on large scale empirical evaluation using other real datasets or realistic synthetic datasets with different characteristics is required to investigate if our proposed techniques are suitable for different datasets. Such datasets could include records with longer attribute values or attributes other than names and addresses. Investigating efficient and interactive cryptanalysis attacks for privacy evaluation would be another direction for future research. As we discuss in Chapter 11, measuring the practicality, such as ease of implementation, of the proposed techniques is another direction for future work.

Chapter 11

Conclusion and Future Directions

This thesis is the first to present comprehensive research in the area of blocking of multidatabase privacy-preserving record linkage (MD-PPRL). In the previous chapters, we have described the MD-PPRL process and challenges related to blocking in MD-PPRL, and proposed several solutions that address these challenges. In this chapter, we first outline the research problem and challenges we have identified in the area of MD-PPRL with regards to blocking in Sections 11.1 and 11.2, respectively. Then, we summarise our contributions in Section 11.3, and outline possible future directions of research in Section 11.4. Finally, we conclude this thesis in Section 11.5 with a summary.

11.1 Outline of the Research Problem

As we have described in Chapter 1, the linkage of multiple databases potentially requires each record from one database to be compared with all records in the other databases in order to determine the set of records that correspond to the same entity. Scalability, linkage quality, and privacy are the three main challenges that are associated with any privacy-preserving record linkage (PPRL) application [176, 212].

In record linkage (RL) and PPRL, blocking is generally used to scale the linkage of databases that contain large numbers of records [36, 165]. Blocking groups similar records into blocks and ensures dissimilar records are assigned into different blocks. By doing so, blocking reduces the record comparison space to only those comparisons that are likely to be matches (correspond to the same entity) while removing as many non-matching record comparisons as possible.

As we have reviewed in Chapter 3, many blocking techniques have been proposed for linking of two databases [36, 165]. However, most of these techniques are not suitable in PPRL for multiple databases. The naive pair-wise comparison of multiple databases grows quadratically as the databases to be matched get larger, and exponentially as the number of databases to be linked increases. Hence, novel blocking techniques that can scale the MD-PPRL process to large databases are required. These techniques must not compromise the privacy and confidentiality of the records in these databases. We next briefly outline the challenges that are related to blocking in a MD-PPRL context.

11.2 Challenges of blocking in a MD-PPRL context

As we described in Section 1.2, apart from the three key challenges above, blocking in MD-PPRL is challenged by several other reasons.

1. **Number of databases**

   In MD-PPRL, when the number of databases is increasing the naive record comparison space grows exponentially. This makes multidatabase linkage applications currently not scalable with regard to an increasing number of databases.

2. **Database size**

   MD-PPRL generally involves linking databases with large number of records. The techniques used in MD-PPRL should be capable of generating blocks where their average sizes remain small to achieve a highly efficient blocking process.

3. **Data quality**

   In both RL and PPRL, the quality of data could decrease rapidly with the increasing number of databases to be linked, because of the different characteristics (heterogeneity), variations, and erroneous (including missing) values in records. This leads to the effectiveness of the overall linkage process to be reduced. Therefore, the blocking techniques used in MD-PPRL should be capable of generating blocks effectively even in the presence of noisy data in order to provide high blocking quality.

4. **Collusion between participating parties**

   A linkage conducted across databases held by different organisations often raises privacy and confidentiality concerns. In a MD-PPRL application, the collusion risk can increase with the number of participating parties. This requires the blocking techniques deployed in MD-PPRL to be resistant to collusion such that the sensitive information of a non-colluding database owner is protected.

5. **Subgroups structures**

   The subsets of matching records across subgroups of databases are valuable for conducting analytical studies about sub-populations within a large population [41, 131]. In order to identify such subsets, a blocking technique used in a MD-PPRL application should be capable of generating blocks that need to be compared across subgroups of databases.

6. **Redundant record comparisons**

   As we have described in Chapter 1 and experimentally evaluated in Chapter 9, the block comparisons in MD-PPRL can contain redundant (repetitive and superfluous) record pair comparisons that occur due to repetitive block comparisons, as well as comparisons of records with record pairs that have previously been classified as non-matches [164]. Therefore, novel techniques are required to remove these redundant record comparisons efficiently without affecting the effectiveness of the overall linkage.

11.3 Summary of Our Contributions

This section provides a list of the main contributions presented in this thesis:

1. **Scalable blocking techniques for MD-PPRL**

   Based on the identified need for novel efficient blocking techniques that are suitable for a MD-PPRL application we proposed two different techniques that are scalable to an increasing number of databases and their sizes. These proposed approaches are specifically suitable for linkage applications where a linkage unit (LU) is not available.

   - **A tree based blocking technique**
     In Chapter 5 we proposed a tree based blocking approach that uses a single-bit tree data structure for assigning records into blocks [170]. We used Bloom filter (BF) encoding as the privacy technique to encode the records in the databases to be blocked. The aim of this approach is to allow database owners to block their databases collaboratively by only sharing a set of bit positions used for splitting.
   - **A clustering based blocking technique**
     In Chapter 6 we proposed a clustering based blocking technique for MD-PPRL [171], with two variations of a clustering technique that can be used in this approach. The first is based on standard canopy clustering while the second is a novel hierarchical canopy clustering technique. Similar to our tree based blocking approach all the records are encoded into BFs before they are arranged into blocks. We also proposed a novel multi-bit splitting approach to increase the efficiency of blocking.

   According to the experiments conducted with different databases with a large number of records, these proposed approaches are scalable to an increasing number of databases and their sizes. The empirical evaluation showed that the blocks generated by these approaches are secure against the cryptanalysis attack described in Section 4.5, while providing higher reduction ratio and pairs completeness for the linkage of an increasing number of databases even with dirty data.

2. **A distributed blocking approach for MD-PPRL**

   In Chapter 7 we proposed a distributed blocking technique for MD-PPRL [173]. This approach can be utilised in a multidatabase linkage model with a LU. The aim of this approach is to allow database owners (DO) to block their databases independently without any communication. Such blocking provides the DOs with flexibility and control over their blocking process. The LU identifies the block tuples that need to be compared based on block representatives sent to it by the DOs. While this approach allows DOs to use any existing technique for blocking, we proposed a hierarchical clustering approach that uses a splitting and merging technique for generating blocks.

3. **A subgroup blocking approach for MD-PPRL**

   In Chapter 8 we proposed a novel subgroup blocking technique for MD-PPRL. This approach uses a LU to generate the candidate block tuples that need to be compared across different subgroups of databases. Our approach allows the LU to specify the minimum and maximum number of databases that can be included in subgroup combinations. The LU first generates a weighted graph structure based on sets of block description pairs it received by the DOs. The LU then uses this graph to identify candidate block tuples for subgroups of different sizes. The proposed approach uses a depth-first based iterative deepening algorithm for generating candidate block tuples for different subgroups combinations. The evaluation with several large real datasets indicated that our approach is scalable with the size of subgroups and number of databases and improves efficiency of candidate blocks generation with no loss in effectiveness.

4. **A meta-blocking approach to reduce redundant record comparisons**

   In Chapter 9 we proposed an efficient non-parametric meta-blocking technique for MD-PPRL that can be used to remove redundant record comparisons [172]. The aim of the proposed approach is to schedule block pair comparisons such that repeated and superfluous record comparisons are removed from the record comparison space. We proposed four variations of scheduling techniques that are suitable for both multidatabase linkage models described in Section 1.3. The proposed approach uses a weighted graph to identify the ordering of block pairs to be compared. We also proposed five different methods to calculate weights for the edges of this graph. An empirical evaluation study conducted on real-world datasets has shown that our approach can improve the overall efficiency of the linkage by scheduling the block pair comparisons effectively without sacrificing the linkage quality.

5. **A blocking framework for MD-PPRL**

   The contributions presented so far have aimed to address a specific aspect related to blocking in a MD-PPRL application. In Chapter 4 we proposed a framework that incorporates all these techniques. This framework enables PPRL practitioners to work with different techniques and to combine them into a single workflow. This allows them to run these techniques using different parameter settings and investigate the applicability of each technique under different real-world linkage situations.

   As we have described in Chapter 4, the proposed blocking framework follows a layered architecture where each layer is responsible for a specific aspect of blocking in a MD-PPRL context. The goal of this framework is to provide a flexible platform for combining different blocking techniques for the linkage of multiple large databases while guaranteeing either the efficiency or effectiveness (or both) of the linkage is always maximised at each layer without compromising privacy.

6. **A collusion resistant secure multi-party computation protocol**

   The blocking approaches proposed in Chapters 5 and 6 use a secure summation protocol to compute a global summed value securely without compromising the privacy of data held by each database owner. However, the basic secure summation protocol [45] is susceptible to collusion attacks which can compromise the privacy of these blocking techniques. To address this issue, in Chapter 4 we investigated the applicability of several existing secure summation protocols under different collusion scenarios. We consider possible collusions that can occur in both linkage models discussed in Section 1.3. We proposed a novel secure summation protocol in Section 4.3 (on page 74) that provides improved privacy compared to the existing secure summation protocols. The use of such a protocol in our blocking techniques makes them more secure against collusion attacks.

7. **A comprehensive evaluation for the proposed techniques**

   In Chapter 10 we conducted a comprehensive experimental evaluation of our proposed techniques in terms of scalability, blocking quality and privacy using multiple large real-world and synthetic datasets. The aim of our empirical evaluation was to examine if the proposed approaches are suitable for use in MD-PPRL under different linkage scenarios. We compared our proposed approaches with several state-of-the-art blocking techniques [36, 56, 119]. This extensive experimental study has led to a better understanding of the characteristics of different techniques and their applicability in real-world scenarios.

11.4 Future Directions

The research presented in this thesis opens various questions that could be explored in the future. These research questions include are:

1. **Parallelisation of blocking in MD-PPRL**

   Parallelising computations among different computational resources can be used to improve the scalability of MD-PPRL. All the blocking techniques presented in this thesis can be parallelised using distributed computing frameworks such as MapReduce [51]. Several initial works on parallelism in record linkage and PPRL have been proposed [64, 86, 116, 126], however, further investigation is required for multidatabase linkage. Therefore, investigating how parallelism can improve the scalability of our proposed algorithms without compromising privacy is one avenue for future research that could explore further.

2. **Private blocking for other adversarial models**

   One limitation of all our proposed blocking approaches is the assumption of the honest but curious (HBC) adversarial model. Though the HBC model has been widely assumed in PPRL, this model is not sufficient in many real-world applications. In the HBC model we assume all parties are following the steps of our blocking approaches while trying to learn as much as possible about the other parties' data.

However, in real applications, dishonest parties could deviate from our blocking approaches by providing invalid input that could compromise the privacy of the entire linkage protocol. In PPRL limited work has been conducted using the malicious adversarial model that provides strong privacy guarantees [137, 152]. A problem with the malicious model is that it generally requires computationally more expensive privacy techniques. This makes it difficult to be adopted in practice and therefore not suitable for MD-PPRL.

On the other hand, adversary models (described in Section 2.4 on page 27) such as covert and accountable computing provide an appropriate level of privacy even when the participating parties behave dishonestly [10, 103, 105]. The covert model guarantees that the participating parties can identify the dishonest behaviour of another party with high probability [10], while the accountable computing model provides accountability for privacy compromises by a party without excessive complexity and cost compared to the malicious model [105]. Therefore, extending the proposed blocking techniques in this thesis into these adversary models and proving the privacy of solutions under these models will require further research.

3. **Blocking for dynamic and temporal MD-PPRL**

   All the blocking techniques presented in this thesis are applicable only to static databases where their records do not change over time. However, real applications in organisations, such as banks, online stores, social networks, etc., often generate records with timestamps. In such databases records are potentially updated when attribute values related to entities are changing. For example in a government department, when a person informs the department about changes to his name, address, or phone number then all records relating to this person need to be updated accordingly. These updated records are given a new timestamp in the relevant databases to reflect the changes that occurred in these records.

   In general, databases are changed frequently by new records being inserted or by updates or deletions of existing records. In such situations, the technique used for blocking should be able to adopt to these changes by updating the generated blocks accordingly. This will ensure the records in temporal databases are assigned to blocks not only by considering their attribute values but also based on the temporal constraints. Initial work has recently been proposed in record linkage for linking temporal databases [37, 97], though these techniques are not yet mature enough to be adapted into PPRL. Therefore, developing techniques that can be used to block temporal and dynamic databases while preserving the privacy is an open question that requires more investigation.

4. **Blocking techniques for different data types**

   As we are living in a Big Data era, databases increasingly store records with different data types such as numeric, date, time, geographic, categorical, etc. Developing techniques to incorporate different data types in blocking is beneficial in order to improve the quality of blocking [174, 176]. Several techniques have been proposed in PPRL that can perform linkage upon different data types [90, 211, 216], however, only a few have been proposed for a multidatabase context [91, 119]. The techniques presented in this thesis are capable of blocking records with textual (string) attribute values, however, they cannot be used with other data types. Therefore, further investigation is required to develop techniques (for a multidatabase linkage context) that can block different data types.

5. **Blocking in the presence of missing values**

   In record linkage and PPRL, current techniques compare the attribute values that are generally available in a pair of records to identify a pair as a match or a non-match. In various domains, missing values may be present in records for a variety of reasons, such as unknown or non-existence, not able to be revealed due to privacy reasons, or due to data entry errors. However, in real applications how to conduct an effective linkage in the presence of missing data is a fundamental question that has not be answered yet. Therefore, the blocking technique used in PPRL should be still capable of assigning records with missing values into blocks.

   In statistical analysis, a variety of methods has been developed to address the problems of missing values [87]. In general, the removal of records with missing values in the analysis could be used as a possible solution, but potentially limits the value of the analysis result. Another possible approach would be to impute missing values with the most likely value using a rule or classification based technique. Recently, Chi et al. [31] proposed a technique for linking two databases in the presence of missing values. Their approach uses the attribute values of the nearest neighbours of a record to impute missing values. However, an extension of such an approach to MD-PPRL requires further research because such techniques need to scale to an increasing numbers and sizes of databases.

6. **Efficient filtering techniques for MD-PPRL**

   As we have discussed in Chapter 1, MD-PPRL often consists of redundant record pair comparisons that contain unnecessary computations in the overall linkage process. To overcome this issue, in Chapter 9 we proposed an efficient meta-blocking technique to schedule the block pair comparisons that can reduce the number of repeated and superfluous record pair comparisons. However, the comparison space of MD-PPRL still contains a large number of potential non-matches that requires further filtering before the classification step. Such filtering techniques help to prune potential non-matching record pairs based on their properties (e.g. length differences of their attribute values). As we have reviewed in Section 3.1 on page 41, several filtering techniques have been proposed in PPRL for the linkage of two databases that can efficiently remove non-matching record pairs [195, 196]. However, to extend or adapt these techniques into a MD-PPRL context requires further investigation.

7. **Cryptanalysis methods for MD-PPRL**

   The proposed blocking techniques in this thesis use a generalisation in blocking by ensuring all the generated blocks contain at least the same minimum number of records, which provides k-anonymous privacy [200]. Based on the cryptanalysis method we described in Section 4.5, we showed that it is impossible to re-identify individual attribute values in blocks generated by our approaches. However, a limitation of using this cryptanalysis method is that it does not consider the information shared between the participating parties when analysing the frequency distributions of the attribute values. Hence, the development of cryptanalysis method that can incorporate the information communicated between participating parties is another future direction in MD-PPRL.

8. **An evaluation framework for MD-PPRL**

   This thesis mainly focused on developing a blocking framework for MD-PPRL. As we have described in Chapter 4, our framework allows PPRL practitioners to combine different techniques into a workflow that can block multiple databases efficiently and effectively. However, extending our blocking framework into an overall MD-PPRL framework (a practical linkage toolbox) that also includes techniques for private comparison and classification is another area of future work that needs further research. Such a framework should be able to incorporate different techniques to perform both blocking and classification in MD-PPRL while also providing functions for the main building blocks of PPRL, the required communication protocols, as well as user and key management functionalities.

   Nevertheless, such evaluation framework allows data linkage researches, practitioners, and novice users to work with different linkage techniques, approaches, and protocols to create highly scalable MD-PPRL solutions that can be easily tailored to the particular settings and requirements of a given application. This also enables users to understand and differentiate the capabilities (advantages and disadvantages) of existing linkage techniques under different linkage contexts. Such practical understanding will assist linkage users to use more state-of-the-art techniques on their linkage applications rather depend on traditional techniques and provided with flexibility to select different combinations of techniques suitable for their linkage scenario to maximise scalability, blocking quality, and privacy. Furthermore, such framework needs to be kept freely accessible for any user who is interested in doing record linkage. This would help the linkage techniques that are utilised in this framework to be used widely.

Furthermore, a common standard set of measures to evaluate scalability, linkage quality, and privacy is also required because PPRL techniques developed by various research groups around the world use a variety of metrics in their evaluations [92]. Having such a standard set of measures will enable PPRL practitioners to conduct comparative evaluations more appropriately which ultimately enables them to select the most suitable techniques for their requirements. Nevertheless, to the best of our knowledge, no PPRL literature has considered the practical aspects, such as ease of implementation, complexity of the protocol, requirement of external software, and the dependability of computation platforms, etc., in their evaluations. However, such measures help novice users to understand the practicality of PPRL techniques before these techniques can be deployed in their linkage applications.

Apart from these future research directions, providing a way to calculate optimal parameter settings for our blocking approaches requires further research. Also, extending our blocking approaches to use other encoding techniques (not only Bloom filters) is another area of future work that needs further investigation.

11.5 Conclusion

This thesis is the first to present comprehensive research in the area of multidatabase privacy-preserving record linkage (MD-PPRL). First, based on an extensive review of existing PPRL techniques we have identified several research questions related to blocking in MD-PPRL. To address these identified gaps, we have proposed a scalable framework for blocking in MD-PPRL that can be used to combine different techniques to create an efficient blocking workflow. We proposed several MD-PPRL blocking techniques that are applicable to both multidatabase linkage models that are performed with or without a linkage unit. We also proposed a scalable subgroup blocking technique for MD-PPRL that efficiently generates blocks to be compared across different subgroups. Our proposed meta-blocking techniques improve the efficiency of the overall linkage process by removing repetitive and superfluous record comparisons without sacrificing the linkage quality. We have conducted a comprehensive comparative evaluation of our proposed solutions and several other state-of-the-art blocking solutions with regard to their efficiency, effectiveness, and privacy. The empirical results showed that our approaches are scalable to increasing number of databases and their sizes while achieving better blocking quality in the presence of dirty data. The results also showed that our approaches provide strong privacy against different privacy attacks compared to existing blocking solutions which make our blocking approach more suitable for MD-PPRL applications.

To conclude, the work presented in this thesis provides an insight into the importance of improving the scalability of MD-PPRL by using blocking. We believe that the proposed techniques can be used practically in many real-world multidatabase linkage scenario where blocking is required.

Bibliography

1. Aggarwal, C. C. and Wang, H., 2010. *Managing and Mining Graph Data*. Springer. (cited on pages 162, 163, 173, and 202)

2. Agrawal, R.; Evfimievski, A.; and Srikant, R., 2003. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 86–97. ACM. (cited on pages 33 and 56)

3. Al-Lawati, A.; Lee, D.; and McDaniel, P., 2005. Blocking-aware private record linkage. In *International Workshop on Information Quality in Information Systems*, 59–68. (cited on pages 2, 21, 27, 33, 35, 41, 43, and 101)

4. Allahbakhsh, M.; Ignjatovic, A.; Benatallah, B.; Beheshti, S.-M.-R.; Bertino, E.; and Foo, N., 2013. Collusion detection in online rating systems. In *Web Technologies and Applications*, 196–207. Springer Berlin Heidelberg. (cited on page 35)

5. Alwen, J.; Katz, J.; Lindell, Y.; Persiano, G.; shelat, a.; and Visconti, I., 2009. Collusion-free multiparty computation in the mediated model. In *Advances in Cryptology*, 524–540. (cited on page 35)

6. Arasu, A.; Götz, M.; and Kaushik, R., 2010. On active learning of record matching packages. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 783–794. ACM. (cited on page 23)

7. Arora, S. and Barak, B., 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edn. (cited on page 82)

8. Aslam, J. A.; Pelekhov, E.; and Rus, D., 2004. The star clustering algorithm for static and dynamic information organization. *J. Graph Algorithms Appl.*, 8 (2004), 95–129. (cited on page 58)

9. Atallah, M.; Kerschbaum, F.; and Du, W., 2003. Secure and private sequence comparisons. In *ACM workshop on Privacy in the Electronic Society*, 39–44. (cited on pages 55 and 101)

10. Aumann, Y. and Lindell, Y., 2007. Security against covert adversaries: Efficient protocols for realistic adversaries. In *TCC*. (cited on pages 27, 28, and 220)

11. Aumann, Y. and Lindell, Y., 2008. Efficient two party and multi-party computation against covert adversaries. In *Advances in Cryptology, Springer-Verlag LNCS*, 289–306. (cited on page 28)

12. Bachteler, T.; Reiher, J.; and Schnell, R., 2013. Similarity filtering with multibit trees for record linkage. Technical report, Working Paper WP-GRLC-2013-02, German Record Linkage Center, Nuremberg. (cited on page 49)

13. Bachteler, T.; Schnell, R.; and Reiher, J., 2010. An empirical comparison of approaches to approximate string matching in private record linkage. In *Proceedings of Statistics Canada Symposium 2010. Social Statistics: The Interplay among Censuses, Surveys and Administrative Data*. (cited on page 21)

14. Baxter, R.; Christen, P.; and Churches, T., 2003. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD'03 workshop on Data Cleaning, Record Linkage and Object Consolidation*, 25–27. Washington DC. (cited on page 8)

15. Benjelloun, O.; Garcia-Molina, H.; Menestrina, D.; et al., 2009. Swoosh: a generic approach to entity resolution. *VLDB*, (2009). (cited on page 184)

16. Bilenko, M. and Mooney, R. J., 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 39–48. ACM. (cited on page 23)

17. Bizer, C.; Heath, T.; and Berners-Lee, T., 2009. Linked data-the story so far. *Semantic services, interoperability and web applications: emerging concepts*, (2009), 205–227. (cited on page 25)

18. Bloom, B., 1970. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13, 7 (1970), 422–426. (cited on pages 28, 46, and 68)

19. Bonomi, L.; Xiong, L.; Chen, R.; and Fung, B., 2012. Frequent grams based embedding for privacy preserving record linkage. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, 1597–1601. ACM. (cited on pages 28, 30, and 54)

20. Bosu, A.; Liu, F.; Yao, D. D.; and Wang, G., 2017. Collusive data leak and more: Large-scale threat analysis of inter-app communications. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 71–85. (cited on page 35)

21. Bouzelat, H.; Quantin, C.; and Dusserre, L., 1996. Extraction and anonymity protocol of medical file. In *Proceedings of the AMIA Annual Fall Symposium*, 323. American Medical Informatics Association. (cited on page 30)

22. Boyd, J. H.; Ferrante, A. M.; O'Keefe, C. M.; Bass, A. J.; Randall, S. M.; and Semmens, J. B., 2012. Data linkage infrastructure for cross-jurisdictional health-related research in australia. *BMC health services research*, 12, 1 (2012), 480. (cited on pages 1, 4, 5, 10, 12, 20, 24, 63, 155, and 156)

23. Boyd, J. H.; Randall, S. M.; and Ferrante, A. M., 2015. Application of privacy-preserving techniques in operational record linkage centres. In *Medical Data Privacy Handbook*, 267–287. Springer. (cited on pages 20, 22, 24, and 25)

24. Boyd, J. H.; Randall, S. M.; Ferrante, A. M.; Bauer, J. K.; McInneny, K.; Brown, A. P.; Spilsbury, K.; Gillies, M.; and Semmens, J. B., 2015. Accuracy and completeness of patient pathways–the benefits of national data linkage in australia. *BMC health services research*, 15, 1 (2015), 312. (cited on pages 1, 5, 20, and 156)

25. Broder, A., 1997. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, 21–29. (cited on pages 45, 137, 141, 142, 145, 163, and 180)

26. Broder, A.; Mitzenmacher, M.; and Mitzenmacher, A., 2002. Network applications of bloom filters: A survey. In *Internet Mathematics*. Citeseer. (cited on pages 28 and 68)

27. Brown, A.; Borgs, C.; Randall, S.; and Schnell, R., 2017. High quality linkage using multibit trees for privacy-preserving blocking. *International Journal for Population Data Science*, 1, 1 (2017). (cited on page 49)

28. Brown, A.; Randall, S.; Ferrante, A.; Semmens, J.; and Boyd, J., 2017. Estimating parameters for probabilistic linkage of privacy-preserved datasets. *BMC Medical Research Methodology*, 17, 1 (2017), 95. (cited on pages 19, 25, and 29)

29. Brown, A. P.; Ferrante, A. M.; Randall, S. M.; Boyd, J. H.; and Semmens, J. B., 2017. Ensuring privacy when integrating patient-based datasets: New methods and developments in record linkage. *Frontiers in Public Health*, 5, 34 (2017). (cited on pages 4, 5, and 20)

30. Cao, J.; Rao, F.-Y.; Bertino, E.; and Kantarcioglu, M., 2015. A hybrid private record linkage scheme: separating differentially private synopses from matching records. In *IEEE ICDE*, 1011–1022. IEEE. (cited on pages 30 and 45)

31. Chi, Y.; Hong, J.; Jurek, A.; Liu, W.; and O'Reilly, D., 2017. Privacy preserving record linkage in the presence of missing values. *Information Systems*, (2017). (cited on pages 47 and 221)

32. Chierichetti, F.; Dalvi, N.; and Kumar, R., 2014. Correlation clustering in mapreduce. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 641–650. ACM. (cited on page 58)

33. Christen, P., 2006. A comparison of personal name matching: Techniques and practical issues. In *Workshop on Mining Complex Data, held at IEEE ICDM'06*. Hong Kong. (cited on page 23)

34. Christen, P., 2008. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *ACM SIGKDD'08*, 151–159. Las Vegas. (cited on page 23)

35. Christen, P., 2012. *Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer. (cited on pages 1, 2, 7, 19, 20, 23, 24, 28, 38, 42, 43, 44, 49, 53, 55, 61, 67, 79, 84, 124, 147, 156, 157, 159, 160, 163, 167, 168, 177, 180, 182, 183, and 200)

36. Christen, P., 2012. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE TKDE*, 24, 9 (2012), 1537–1555. (cited on pages 2, 8, 11, 15, 20, 21, 23, 36, 37, 41, 62, 82, 89, 147, 200, 215, and 219)

37. Christen, P. and Gayler, R. W., 2013. Adaptive temporal entity resolution on dynamic databases. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 558–569. Springer. (cited on page 220)

38. Christen, P. and Goiser, K., 2007. Quality and complexity measures for data linkage and deduplication. In *Quality Measures in Data Mining*, vol. 43 of *Studies in Computational Intelligence*, 127–151. Springer. (cited on pages 2, 38, and 83)

39. Christen, P.; Rainer, S.; Vatsalan, D.; and Thilina, R., 2017. Efficient cryptanalysis of bloom filters for privacy-preserving record linkage. In *PAKDD*. Soeul, South Korea. (cited on pages 27, 33, 34, 69, 85, 86, 101, and 207)

40. Christen, P. and Vatsalan, D., 2013. Flexible and extensible generation and corruption of personal data. In *ACM CIKM*, 1165–1168. San Francisco. (cited on pages 80 and 105)

41. Christen, P.; Vatsalan, D.; and Fu, Z., 2015. Advanced record linkage methods and privacy aspects for population reconstruction—a survey and case studies. In *Population Reconstruction* (Eds. G. Bloothooft; P. Christen; K. Mandemakers; and M. Schraagen), 87–110. Springer. (cited on pages 5, 10, 156, and 216)

42. Churches, T. and Christen, P., 2004. Some methods for blindfolded record linkage. *BioMed Central Medical Informatics and Decision Making*, 4, 9 (2004). (cited on pages 2, 21, and 54)

43. Churches, T.; Christen, P.; Lim, K.; and Zhu, J. X., 2002. Preparation of name and address data for record linkage using hidden Markov models. *BioMed Central Medical Informatics and Decision Making*, 2, 9 (2002). (cited on page 22)

44. Clifton, C.; Kantarcioglu, M.; Doan, A.; Schadow, G.; Vaidya, J.; Elmagarmid, A.; and Suciu, D., 2004. Privacy-preserving data integration and sharing. In *Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, 19–26. ACM. (cited on pages 4 and 34)

45. Clifton, C.; Kantarcioglu, M.; Vaidya, J.; Lin, X.; and Zhu, M., 2002. Tools for privacy preserving distributed data mining. *SIGKDD Explorations*, 4, 2 (2002), 28–34. (cited on pages 30, 32, 70, and 219)

46. Cohen, E.; Datar, M.; Fujiwara, S.; Gionis, A.; Indyk, P.; Motwani, R.; et al., 2001. Finding interesting associations without support pruning. *IEEE TKDE*, (2001). (cited on page 143)

47. Cohen, W. W.; Ravikumar, P.; and Fienberg, S., 2003. A comparison of string distance metrics for name-matching tasks. In *Workshop on Information Integration on the Web, held at IJCAI'03*. Acapulco. (cited on page 23)

48. Cohen, W. W. and Richman, J., 2002. Learning to match and cluster large high-dimensional data sets for data integration. In *ACM SIGKDD'02*, 475–480. Edmonton. (cited on pages 41, 113, and 119)

49. Cormode, G., 2009. Count-min sketch. In *Encyclopedia of Database Systems*, 511–516. Springer. (cited on page 29)

50. Cramton, P. and Schwartz, J. A., 2000. Collusive bidding: Lessons from the fcc spectrum auctions. *Journal of Regulatory Economics*, 17, 3 (2000), 229–252. (cited on page 35)

51. Dean, J. and Ghemawat, S., 2008. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51, 1 (2008), 107–113. (cited on pages 53 and 219)

52. Dong, B.; Liu, R.; and Wang, W. H., 2014. Prada: Privacy-preserving data-deduplication-as-a-service. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 1559–1568. ACM. (cited on page 34)

53. Dong, C.; Chen, L.; and Wen, Z., 2013. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security*, 789–800. (cited on pages 29, 30, and 33)

54. Duncan, G. T. and Elliot, M. J., 2011. *Statistical confidentiality: principles and practice*. Springer. (cited on pages 39 and 81)

55. Dunn, H., 1946. Record linkage. *American Journal of Public Health*, 36, 12 (1946), 1412. (cited on page 19)

56. Durham, E., 2012. *A framework for accurate, efficient private record linkage*. Ph.D. thesis, Faculty of the Graduate School of Vanderbilt University, Nashville, TN. (cited on pages 15, 34, 38, 41, 43, 45, 46, 62, 63, 69, 141, 144, 167, 191, 200, 201, and 219)

57. Durham, E.; Xue, Y.; Kantarcioglu, M.; and Malin, B., 2010. Private medical record linkage with approximate matching. In *AMIA Annual Symposium Proceedings*, vol. 2010, 182. American Medical Informatics Association. (cited on page 5)

58. Durham, E.; Xue, Y.; Kantarcioglu, M.; and Malin, B., 2012. Quantifying the correctness, computational complexity, and security of privacy-preserving string comparators for record linkage. *Information Fusion*, 13, 4 (2012), 245–259. (cited on page 2)

59. Durham, E. A.; Toth, C.; Kuzu, M.; Kantarcioglu, M.; Xue, Y.; and Malin, B., 2013. Composite bloom filters for secure record linkage. *IEEE Transactions on Knowledge and Data Engineering*, (2013). (cited on pages 29, 43, 68, 79, and 101)

60. Durstenfeld, R., 1964. Algorithm 235: Random permutation. *Commun. ACM*, 7, 7 (1964), 420–. (cited on page 46)

61. DuVall, S. L.; Kerber, R. A.; and Thomas, A., 2010. Extending the fellegi–sunter probabilistic record linkage method for approximate field comparators. *Journal of biomedical informatics*, 43, 1 (2010), 24–30. (cited on page 19)

62. Dwork, C., 2006. Differential privacy. *International Colloquium on Automata, Languages and Programming*, (2006), 1–12. (cited on page 29)

63. Dwork, C.; Roth, A.; et al., 2014. The algorithmic foundations of differential privacy. *Foundations and Trends ® in Theoretical Computer Science*, 9, 3–4 (2014), 211–407. (cited on page 30)

64. Efthymiou, V.; Papadakis, G.; Papastefanatos, G.; Stefanidis, K.; and Palpanas, T., 2017. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Information Systems*, 65 (2017), 137–157. (cited on pages 53 and 219)

65. Elgamal, T., 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31, 4 (1985), 469–472. (cited on page 76)

66. Elmagarmid, A.; Ipeirotis, P.; and Verykios, V. S., 2007. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19, 1 (2007), 1–16. (cited on pages 1, 2, 4, 12, 19, 20, 23, and 26)

67. Esayas, S. Y., 2015. The role of anonymisation and pseudonymisation under the eu data privacy rules: beyond the âĂŸall or nothingâĂŹ approach. *European Journal of Law and Technology*, 6, 2 (2015), 1–23. (cited on page 21)

68. Etienne, B.; Cheatham, M.; and Grzebala, P., 2016. An analysis of blocking methods for private record linkage. In *2016 AAAI Fall Symposium Series*. (cited on page 49)

69. Fan, L.; Cao, P.; Almeida, J.; and Broder, A. Z., 2000. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking (TON)*, 8, 3 (2000), 281–293. (cited on page 29)

70. Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., 1996. *Advances in knowledge discovery and data mining*. American Association for Artificial Intelligence. (cited on page 1)

71. Fellegi, I. P. and Sunter, A. B., 1969. A theory for record linkage. *Journal of the American Statistical Society*, 64, 328 (1969). (cited on pages 1, 11, 19, 20, 23, 41, and 200)

72. Ferrante, A., 2009. The use of data-linkage methods in criminal justice research: A commentary on progress, problems and future possibilities. *Current Issues in Criminal Justice*, 20, 03 (2009), 378–392. (cited on page 5)

73. Fienberg, S. E. and Manrique-Vallier, D., 2009. Integrated methodology for multiple systems estimation and record linkage using a missing data formulation. *AStA Advances in Statistical Analysis*, 93, 1 (2009), 49–60. (cited on page 82)

74. Flack, F.; Wray, N.; and Smith, M., 2017. The impact of the population health research network on linked data research capacity in australia. *International Journal for Population Data Science*, 1, 1 (2017). (cited on page 1)

75. for National Statistics, O., 2013. Beyond 2011: Matching anonymous data. http://www.ons.gov.uk/ons/about-ons/who-ons-are/programmes-and-projects/beyond-2011/reports-and-publications/methods-and-policies-reports/index.html. Accessed: 2017-01-20. (cited on page 20)

76. Freedman, M.; Ishai, Y.; Pinkas, B.; and Reingold, O., 2005. Keyword search and oblivious pseudorandom functions. *Theory of Cryptography*, (2005), 303–324. (cited on pages 31 and 55)

77. Fu, Z., 2014. *Linking Historical Census Data Across Time*. Ph.D. thesis, The Australian National University, Canberra, Australia. (cited on pages 5, 20, and 80)

78. Fu, Z.; Christen, P.; and Zhou, J., 2014. A graph matching method for historical census household linkage. In *PAKDD*. (cited on page 5)

79. Fung, B.; Wang, K.; Chen, R.; and Yu, P., 2010. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, (2010). (cited on page 30)

80. Ganta, S. R.; Kasiviswanathan, S. P.; and Smith, A., 2008. Composition attacks and auxiliary information in data privacy. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 265–273. ACM, New York, NY, USA. (cited on page 34)

81. Gentry, C., 2009. *A Fully Homomorphic Encryption Scheme*. Ph.D. thesis, Stanford University, Stanford, CA, USA. (cited on page 31)

82. Gill, L., 2001. Methods for automatic record matching and linking and their use in national statistics. Technical Report Methodology Series, no. 25, National Statistics, London. (cited on page 20)

83. Goldreich, O., 2002. Secure multi-party computation. Technical report, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Israel. (cited on page 27)

84. Goldreich, O.; Micali, S.; and Wigderson, A., 1987. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, 218–229. ACM. (cited on page 30)

85. Gu, L. and Baxter, R., 2006. Decision models for record linkage. In *Data Mining*, 146–160. Springer. (cited on page 23)

86. Guisado-Gámez, J.; Prat-Pérez, A.; Nin, J.; Muntés-Mulero, V.; and Larriba-Pey, J. L., 2008. Parallelizing record linkage for disclosure risk assessment. In *Privacy in Statistical Databases*, 190–202. Springer. (cited on page 219)

87. Hair, J. F.; Black, W. C.; Babin, B. J.; Anderson, R. E.; Tatham, R. L.; et al., 1998. *Multivariate data analysis*, vol. 5. Prentice hall Upper Saddle River, NJ. (cited on page 221)

88. Hall, R. and Fienberg, S., 2010. Privacy-preserving record linkage. In *Privacy in Statistical Databases, Springer LNCS 6344*, 269–283. Corfu, Greece. (cited on pages 2, 19, 21, 27, and 31)

89. Han, J.; Kamber, M.; and Pei, J., 2011. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edn. (cited on page 57)

90. Han, S.; Shen, D.; Nie, T.; Kou, Y.; and Yu, G., 2016. Scalable private blocking technique for privacy-preserving record linkage. In *Asia-Pacific Web Conference*, 201–213. Springer. (cited on pages 47, 50, 79, and 221)

91. Han, S.; Shen, D.; Nie, T.; Kou, Y.; and Yu, G., 2017. Private blocking technique for multi-party privacy-preserving record linkage. *Data Science and Engineering*, 2, 2 (2017), 187–196. (cited on pages 50, 79, 201, and 221)

92. Hand, D. and Christen, P., 2017. A note on using the f-measure for evaluating record linkage algorithms. *Statistics and Computing*, (2017). (cited on pages 37 and 223)

93. Hassanzadeh, O. and Miller, R. J., 2009. Creating probabilistic databases from duplicated data. *VLDB*, 18, 5 (2009), 1141–1166. (cited on page 58)

94. Hawashin, B.; Fotouhi, F.; and Truta, T., 2011. A privacy preserving efficient protocol for semantic similarity join using long string attributes. In *Proceedings of the 4th International Workshop on Privacy and Anonymity in the Information Society*, 6. ACM. (cited on pages 19, 92, 115, and 136)

95. Hernandez, M. A. and Stolfo, S. J., 1998. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2, 1 (1998), 9–37. (cited on pages 3, 9, 30, 43, 50, and 53)

96. Herzog, T.; Scheuren, F.; and Winkler, W., 2007. *Data quality and record linkage techniques*. Springer Verlag. (cited on pages 1, 3, and 9)

97. Hu, Y.; Wang, Q.; Vatsalan, D.; and Christen, P., 2017. Improving temporal record linkage using regression classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 561–573. Springer. (cited on page 220)

98. Inan, A.; Kantarcioglu, M.; Bertino, E.; and Scannapieco, M., 2008. A hybrid approach to private record linkage. In *IEEE International Conference on Data Engineering*, 496–505. (cited on pages 21 and 42)

99. Inan, A.; Kantarcioglu, M.; Ghinita, G.; and Bertino, E., 2010. Private record matching using differential privacy. In *International Conference on Extending Database Technology*, 123–134. ACM. (cited on pages 26, 30, 46, and 48)

100. Inan, A.; Kaya, S. V.; Saygin, Y.; Savaş, E.; Hintoğlu, A. A.; and Levi, A., 2007. Privacy preserving clustering on horizontally partitioned data. *Data & Knowledge Engineering*, 63, 3 (2007), 646–666. (cited on page 31)

101. Indyk, P. and Motwani, R., 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Theory of Computing*. (cited on pages 41, 63, 137, 144, 145, 160, 161, and 180)

102. Inokuchi, A.; Washio, T.; and Motoda, H., 2000. An Apriori-based algorithm for mining frequent substructures from graph data. In *ACM PKDD*. (cited on pages 202 and 210)

103. Jiang, W. and Clifton, C., 2006. Ac-framework for privacy-preserving collaboration. Technical Report 06-015, Department of COmpute Science, Purdue University. (cited on pages 28 and 220)

104. Jiang, W. and Clifton, C., 2006. A secure distributed framework for achieving k-anonymity. *The International Journal on Very Large Data Bases*, 15, 4 (2006), 316–333. (cited on page 57)

105. Jiang, W.; Clifton, C.; and Kantarcioglu, M., 2008. Transforming semi-honest protocols to ensure accountability. *Data & Knowledge Engineering*, 65, 1 (2008), 57–74. (cited on page 220)

106. Jonas, J. and Harper, J., 2006. Effective counterterrorism and the limited role of predictive data mining. *Policy Analysis*, (2006). (cited on page 20)

107. Kantarcioglu, M.; Inan, A.; Jiang, W.; and Malin, B., 2009. Formal anonymity models for efficient privacy-preserving joins. *Data & Knowledge Engineering*, 68, 11 (2009), 1206–1223. (cited on pages 29 and 31)

108. Kantarcioglu, M.; Jiang, W.; and Malin, B., 2008. A privacy-preserving framework for integrating person-specific databases. In *Privacy in Statistical Databases*, 298–314. Springer. (cited on pages 26, 31, 57, and 61)

109. Karakasidis, A.; Koloniari, G.; and Verykios, V. S., 2015. Scalable blocking for privacy preserving record linkage. In *ACM KDD*. (cited on pages 41, 46, 53, 61, and 79)

110. Karakasidis, A. and Verykios, V., 2011. Secure blocking+ secure matching= secure record linkage. *Journal of Computing Science and Engineering*, 5, 3 (2011), 223–235. (cited on pages 38, 41, 42, 48, 112, 124, and 200)

111. Karakasidis, A. and Verykios, V., 2012. Reference table based k-anonymous private blocking. In *ACM Symposium on Applied Computing*. Riva del Garda, Italy. (cited on pages 25, 29, 42, 43, 45, 46, 50, 61, and 62)

112. Karakasidis, A.; Verykios, V.; and P., C., 2011. Fake injection strategies for private phonetic matching. In *Submitted to the International Workshop on Data Privacy Management*. Leuven, Belgium. (cited on pages 38 and 42)

113. Karakasidis, A. and Verykios, V. S., 2012. A sorted neighborhood approach to multidimensional privacy preserving blocking. In *IEEE ICDMW*, 937–944. (cited on pages 43, 46, and 50)

114. Karapiperis, D.; Vatsalan, D.; Verykios, V. S.; and Christen, P., 2015. Large-scale multi-party counting set intersection using a space efficient global synopsis. In *DASFAA*. (cited on pages 25, 29, and 58)

115. Karapiperis, D.; Vatsalan, D.; Verykios, V. S.; and Christen, P., 2016. Efficient record linkage using a compact hamming space. In *EDBT*, 209–220. (cited on pages 28 and 50)

116. Karapiperis, D. and Verykios, V. S., 2013. A distributed framework for scaling up lsh-based computations in privacy preserving record linkage. In *Proceedings of the 6th Balkan Conference in Informatics*, 102–109. ACM. (cited on pages 25, 161, and 219)

117. Karapiperis, D. and Verykios, V. S., 2014. A distributed near-optimal lsh-based framework for privacy-preserving record linkage. *Comput. Sci. Inf. Syst.*, 11, 2 (2014), 745–763. (cited on pages 45 and 63)

118. Karapiperis, D. and Verykios, V. S., 2015. An lsh-based blocking approach with a homomorphic matching technique for privacy-preserving record linkage. *TKDE*, (2015). (cited on pages 30, 31, 41, 45, 61, and 144)

119. Karapiperis, D. and Verykios, V. S., 2016. A fast and efficient hamming LSH-based scheme for accurate linkage. *Knowledge and Information Systems*, 49, 3 (2016), 861–884. (cited on pages 15, 25, 46, 50, 61, 79, 200, 201, 219, and 221)

120. Karapiperis, D.; Verykios, V. S.; Katsiri, E.; and Delis, A., 2016. A tutorial on blocking methods for privacy-preserving record linkage. In *Algorithmic Aspects of Cloud Computing*, 3–15. Springer. (cited on page 25)

121. Karmel, R.; Anderson, P.; Gibson, D.; Peut, A.; Duckett, S.; and Wells, Y., 2010. Empirical aspects of record linkage across multiple data sets using statistical linkage keys: the experience of the piac cohort study. *BMC health services research*, 10, 1 (2010), 41. (cited on page 1)

122. Karr, A. F.; Lin, X.; Sanil, A. P.; and Reiter, J. P., 2004. Analysis of integrated data without data integration. *CHANCE*, 17 (2004), 26âĂŞ–29. (cited on page 70)

123. Kelman, C. W.; Bass, J.; and Holman, D., 2002. Research use of linked health data – A best practice protocol. *Aust NZ Journal of Public Health*, 26 (2002), 251–255. (cited on page 20)

124. Kim, H.-S. and Lee, D., 2010. Harra: Fast iterative hashed record linkage for large-scale data collections. In *ACM EDBT*. (cited on page 52)

125. Kissner, L. and Song, D., 2005. Privacy-preserving set operations. In *Proceedings of the 25th Annual International Conference on Advances in Cryptology*, 241–257. Springer-Verlag, Berlin, Heidelberg. (cited on page 33)

126. Kolb, L.; Thor, A.; and Rahm, E., 20102. Dedoop: Efficient deduplication with hadoop. *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases*, (20102). (cited on page 219)

127. Kong, C.; Gao, M.; Xu, C.; Qian, W.; and Zhou, A., 2016. Entity matching across multiple heterogeneous data sources. In *International Conference on Database Systems for Advanced Applications*, 133–146. Springer. (cited on pages 19, 23, 26, and 61)

128. Krawczyk, H.; Bellare, M.; and Canetti, R., 1997. HMAC: Keyed-hashing for message authentication. In *Internet RFCs*. RFC Editor, United States. (cited on pages 30 and 34)

129. Kristensen, T. G.; Nielsen, J.; and Pedersen, C. N., 2010. A tree-based method for the rapid screening of chemical fingerprints. *Algorithms for Molecular Biology*, 5, 1 (2010), 9. (cited on pages 49 and 93)

130. Kroll, M. and Steinmetzer, S., 2015. Who is 1011011111...1110110010? Automated cryptanalysis of Bloom filter encryptions of databases with several personal identifiers. In *BIOSTEC*. Lisbon. (cited on page 34)

131. Kum, H. C.; Krishnamurthy, A.; Machanavajjhala, A.; and Ahalt, S. C., 2014. Social genome: Putting big data to work for population informatics. *Computer*, (2014). (cited on pages 5, 10, 63, 156, and 216)

132. Kuzu, M.; Kantarcioglu, M.; Durham, E.; and Malin, B., 2011. A constraint satisfaction cryptanalysis of Bloom filters in private record linkage. In *Privacy Enhancing Technologies*, 226–245. Springer. (cited on pages 27, 34, 101, 141, 167, and 191)

133. Kuzu, M.; Kantarcioglu, M.; Durham, E. A.; Toth, C.; and Malin, B., 2013. A practical approach to achieve private medical record linkage in light of public resources. *Journal of the American Medical Informatics Association*, 20, 2 (2013), 285. (cited on page 34)

134. Kuzu, M.; Kantarcioglu, M.; Inan, A.; Bertino, E.; Durham, E.; and Malin, B., 2013. Efficient privacy-aware record integration. In *ACM EDBT*. (cited on pages 30, 39, 41, 50, 69, and 112)

135. Lai, P.; Yiu, S.; Chow, K.; Chong, C.; and Hui, L., 2006. An Efficient Bloom Filter Based Solution for Multiparty Private Matching. In *Proceedings of The 2006 International Conference on Security and Management*. (cited on pages 26, 56, 57, and 58)

136. Lavrač, N.; Flach, P.; and Zupan, B., 1999. Rule evaluation measures: A unifying view. *Inductive Logic Programming*, (1999), 174–185. (cited on page 37)

137. Li, F.; Chen, Y.; Luo, B.; Lee, D.; and Liu, P., 2011. Privacy preserving group linkage. In *Scientific and Statistical Database Management*, 432–450. Springer. (cited on pages 26, 27, 55, and 220)

138. Li, J.; Baig, M. M.; Sattar, A. S.; Ding, X.; Liu, J.; and Vincent, M. W., 2016. A hybrid approach to prevent composition attacks for independent data releases. *Information Sciences*, 367 (2016), 324–336. (cited on page 34)

139. Lin, K.-J.; Zou, J.; and Wang, Y., 2010. Accountability computing for e-society. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, 34–41. IEEE. (cited on page 28)

140. Lindell, Y. and Pinkas, B., 2009. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1, 1 (2009), 5. (cited on pages 27, 28, 32, 35, and 148)

141. Liu, H.; Wang, H.; and Chen, Y., 2010. Ensuring data storage security against frequency-based attacks in wireless networks. In *Proceedings of the 6th IEEE International Conference on Distributed Computing in Sensor Systems*, 201–215. Springer-Verlag, Berlin, Heidelberg. (cited on page 33)

142. Lutz, M. and Van, G., 2001. *Programming Python: Object-Oriented Scripting*. O'Reilly & Associates, Inc. (cited on pages 16 and 84)

143. Lyons, R.; Ford, D.; Moore, L.; and Rodgers, S., 2017. Use of data linkage to measure the population health effect of non-health-care interventions. *The Lancet*, 383, 9927 (2017), 1517–1519. (cited on page 25)

144. McCallum, A.; Nigam, K.; and Ungar, L. H., 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *ACM SIGKDD'00*, 169–178. Boston. (cited on pages 41, 113, and 119)

145. Mehnaz, S.; Bellala, G.; and Bertino, E., 2017. A secure sum protocol and its application to privacy-preserving multi-party analytics. In *Proceedings of the 22Nd ACM on Symposium on Access Control Models and Technologies*, 219–230. (cited on page 76)

146. Mena, J., 2003. *Investigative data mining for security and criminal detection*. Butterworth-Heinemann. (cited on pages 6 and 7)

147. Méray, N.; Reitsma, J. B.; Ravelli, A. C.; and Bonsel, G. J., 2007. Probabilistic record linkage is a valid and transparent tool to combine databases without a patient identification number. *Journal of clinical epidemiology*, 60, 9 (2007), 883–e1. (cited on page 19)

148. Micciancio, D., 2010. A first glimpse of cryptography's holy grail. *Commun. ACM*, 53, 3 (2010), 96–96. (cited on page 33)

149. Mitchell, R. J.; Cameron, C. M.; McClure, R. J.; and Williamson, A. M., 2015. Data linkage capabilities in australia: Practical issues identified by a population health research network 'proof of concept project'. *Australian and New Zealand Journal of Public Health*, 39, 4 (2015), 319–325. (cited on page 12)

150. Mitzenmacher, M. and Upfal, E., 2005. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press. (cited on pages 29 and 69)

151. Mohammed, M. A. and Stevens, A., 2007. The value of administrative databases. *BMJ: British Medical Journal*, 334, 7602 (2007), 1014. (cited on page 1)

152. Mohammed, N.; Fung, B.; and Debbabi, M., 2011. Anonymity meets game theory: secure data integration with malicious participants. *The International Journal on Very Large Databases*, 20, 4 (2011), 567–588. (cited on pages 26, 27, 57, and 220)

153. Motwani, R. and Raghavan, P., 1995. *Randomized Algorithms*. Cambridge Uni Press. (cited on page 143)

154. Naehrig, M.; Lauter, K.; and Vaikuntanathan, V., 2011. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, 113–124. ACM, New York, NY, USA. (cited on page 31)

155. Naumann, F. and Herschel, M., 2010. *An Introduction to Duplicate Detection*. Morgan and Claypool Publishers. (cited on pages 1 and 23)

156. Newcombe, H.; Kennedy, J.; Axford, S.; and James, A., 1959. Automatic linkage of vital records. *Science*, 130, 3381 (1959), 954–959. (cited on page 19)

157. Newcombe, H. B., 1988. *Handbook of record linkage: methods for health and statistical studies, administration, and business*. Oxford University Press, Inc., New York, NY, USA. ISBN 0-19-261732-X. (cited on page 20)

158. Niedermeyer, F.; Steinmetzer, S.; Kroll, M.; and Schnell, R., 2014. Cryptanalysis of basic Bloom filters used for privacy preserving record linkage. *JPC*, 6, 2 (2014), 59–79. (cited on page 34)

159. Nigrini, M. J., 2011. *Identifying Fraud Using Abnormal Duplications within Subsets*, 233–262. John Wiley & Sons, Inc. (cited on page 7)

160. OKeefe, C.; Yung, M.; Gu, L.; and Baxter, R., 2004. Privacy-preserving data linkage protocols. In *ACM Workshop on Privacy in the Electronic Society*, 94–102. (cited on pages 26, 31, 32, 34, and 56)

161. Paillier, P., 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, 223–238. Springer-Verlag, Berlin, Heidelberg. (cited on pages 30, 31, 47, and 71)

162. Pang, C.; Gu, L.; Hansen, D.; and Maeder, A., 2009. Privacy-preserving fuzzy matching using a public reference table. *Intelligent Patient Management*, (2009), 71–89. (cited on pages 54 and 61)

163. Papadakis, G.; Ioannou, E.; Palpanas, T.; Niederee, C.; and Nejdl, W., 2013. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Trans. on Knowl. and Data Eng.*, 25, 12 (2013), 2665–2682. (cited on pages 52 and 63)

164. Papadakis, G.; Papastefanatos, G.; and Koutrika, G., 2014. Supervised metablocking. *VLDB*, (2014). (cited on pages 10, 53, 63, and 216)

165. Papadakis, G.; Svirsky, J.; Gal, A.; and Palpanas, T., 2016. Comparative analysis of approximate blocking techniques for entity resolution. *VLDB Endow.*, (2016). (cited on pages 8, 23, 41, 62, 89, 157, and 215)

166. Perl, H.; Mohammed, Y.; Brenner, M.; and Smith, M., 2012. Fast confidential search for bio-medical data using bloom filters and homomorphic cryptography. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, 1–8. IEEE. (cited on page 29)

167. Phua, C.; Smith-Miles, K.; Lee, V.; and Gayler, R., 2012. Resilient identity crime detection. *IEEE Transactions on Knowledge and Data Engineering*, 24, 3 (2012), 533–546. (cited on page 7)

168. Rabin, M. O., 1981. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University. (cited on page 31)

169. Rahm, E. and Do, H. H., 2000. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23, 4 (2000). (cited on page 22)

170. Ranbaduge, T.; Vatsalan, D.; and Christen, P., 2014. Tree based scalable indexing for multi-party privacy-preserving record linkage. In *AusDM, CRPIT 158*. Brisbane. (cited on pages 9, 23, 27, and 217)

171. Ranbaduge, T.; Vatsalan, D.; and Christen, P., 2015. Clustering-based scalable indexing for multi-party privacy-preserving record linkage. In *PAKDD'09, Springer LNAI*. Vietnam. (cited on page 217)

172. Ranbaduge, T.; Vatsalan, D.; and Christen, P., 2016. Scalable block scheduling for efficient multi-database record linkage. In *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, 1161–1166. (cited on page 218)

173. Ranbaduge, T.; Vatsalan, D.; Christen, P.; and Verykios, V. S., 2016. Hashing-based distributed multi-party blocking for privacy-preserving record linkage. In *Advances in Knowledge Discovery and Data Mining - 20th Pacific-Asia Conference, PAKDD 2016, Auckland, New Zealand, April 19-22, 2016, Proceedings, Part II*, 415–427. (cited on pages 156 and 217)

174. Randall, S.; Ferrante, A.; Boyd, J.; and Semmens, J., 2013. The effect of data cleaning on record linkage quality. *BMC Med Inform Decis Mak*, (2013). (cited on pages 3, 22, 79, 184, and 221)

175. Randall, S. M.; Brown, A. P.; Ferrante, A. M.; Boyd, J. H.; and Semmens, J. B., 2015. Privacy preserving record linkage using homomorphic encryption. In *Population Informatics for Big Data*. Sydney, Australia. (cited on page 31)

176. Randall, S. M.; Ferrante, A. M.; Boyd, J. H.; Bauer, J. K.; and Semmens, J. B., 2014. Privacy-preserving record linkage on large real world datasets. *Journal of Biomedical Informatics*, 50, 0 (2014), 205–212. (cited on pages 2, 12, 23, 24, 25, 30, 191, 215, and 221)

177. Randall, S. M.; Ferrante, A. M.; Boyd, J. H.; Brown, A. P.; and Semmens, J. B., 2016. Limited privacy protection and poor sensitivity. *Health Information Management Journal*, 45, 2 (2016), 71–79. (cited on pages 2 and 3)

178. Ravikumar, P. and Cohen, W. W., 2004. A hierarchical graphical model for record linkage. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, 454–461. AUAI Press. (cited on page 19)

179. Reddy, M.; Prasad, B. E.; Reddy, P.; and Gupta, A., 1994. A methodology for integration of heterogeneous databases. *IEEE transactions on knowledge and data engineering*, 6, 6 (1994), 920–933. (cited on page 1)

180. Reich, B. J. and Porter, M. D., 2015. Partially supervised spatiotemporal clustering for burglary crime series identification. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 178, 2 (2015), 465–480. (cited on page 5)

181. Ruggles, S., 2002. Linking historical censuses: A new approach. *History and Computing*, 14, 1-2 (2002), 213–224. (cited on page 20)

182. Russell, S. and Norvig, P., 2009. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press. (cited on page 163)

183. Sadinle, M. and Fienberg, S. E., 2013. A generalized fellegi–sunter framework for multiple record linkage with application to homicide record systems. *Journal of the American Statistical Association*, 108, 502 (2013), 385–397. (cited on pages 8, 19, 23, 82, and 156)

184. Sadinle, M.; Hall, R.; and Fienberg, S. E., 2011. Approaches to multiple record linkage. In *Proceedings of the 58th World Statistical Congress*, 1064–1071. International Statistical Institute, International Statistical Institute, Dublin. (cited on pages 11, 19, 82, and 156)

185. Saeedi, A.; Peukert, E.; and Rahm, E., 2017. Comparative evaluation of distributed clustering schemes for multi-source entity resolution. In *Advances in Databases and Information Systems-21th East European Conference, ADBIS*. (cited on page 58)

186. Scannapieco, M.; Figotin, I.; Bertino, E.; and Elmagarmid, A., 2007. Privacy preserving schema and data matching. In *International conference on Management of Data*, 653–664. ACM. (cited on pages 28, 29, 42, 54, 55, 101, and 135)

187. Schneier, B., 1996. *Applied cryptography: Protocols, algorithms, and source code in C, 2nd Edition*. John Wiley & Sons, Inc., New York. (cited on page 30)

188. Schnell, R., 2014. An efficient privacy-preserving record linkage technique for administrative data and censuses. *Journal of the International Association for Official Statistics*, 30, 3 (2014), 263–270. (cited on pages 5, 49, 92, and 156)

189. Schnell, R., 2015. Privacy-preserving record linkage. In *Methodological Developments in Data Linkage* (Eds. K. Harron; H. Goldstein; and C. Dibben), 201–225. John Wiley & Sons, Inc., UK. (cited on pages 2, 21, 22, 30, 49, 68, and 72)

190. Schnell, R.; Bachteler, T.; and Reiher, J., 2009. Privacy-preserving record linkage using bloom filters. *BMC Medical Informatics and Decision Making*, 9, 1 (2009). (cited on pages 29, 30, 34, 42, 52, 68, 69, 160, 163, and 180)

191. Schnell, R. and Borgs, C., 2015. Building a national perinatal data base without the use of unique personal identifiers. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*, 232–239. IEEE. (cited on page 1)

192. Schnell, R. and Borgs, C., 2016. Randomized response and balanced bloom filters for privacy preserving record linkage. In *Data Mining Workshops (ICDMW), 2016 IEEE 16th International Conference on*, 218–224. IEEE. (cited on pages 29 and 49)

193. Schnell, R. and Borgs, C., 2016. Randomized response and balanced bloom filters for privacy preserving record linkage. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, 218–224. (cited on pages 68 and 85)

194. Schnell, R.; Richter, A.; and Borgs, C., 2017. A comparison of statistical linkage keys with bloom filter-based encryptions for privacy-preserving record linkage using real-world mammography data. In *Proceedings of the 10th International Joint Conference on Biomedical Engineering Systems and Technologies - Volume 5: HEALTHINF, (BIOSTEC 2017)*, 276–283. (cited on pages 21, 29, and 49)

195. Sehili, Z.; Kolb, L.; Borgs, C.; Schnell, R.; and Rahm, E., 2015. Privacy preserving record linkage with ppjoin. In *16th conference on "Database Systems for Business, Technology, and Web*. Hamburg, Germany. (cited on pages 23, 51, 52, and 222)

196. Sehili, Z. and Rahm, E., 2016. Speeding up privacy preserving record linkage for metric space similarity measures. *Datenbank-Spektrum*, 16, 3 (2016), 227–236. (cited on pages 24, 51, 52, and 222)

197. Shannon, C. and Weaver, W., 1962. *The mathematical theory of communication*, vol. 19. University of Illinois Press Urbana. (cited on page 38)

198. Song, D.; Wagner, D.; and Perrig, A., 2000. Practical techniques for searches on encrypted data. *IEEE Symposium of Security and Privacy*, (2000), 0044. (cited on page 48)

199. Steorts, R. C.; Ventura, S. L.; Sadinle, M.; and Fienberg, S. E., 2014. A comparison of blocking methods for record linkage. In *International Conference on Privacy in Statistical Databases*, 253–268. Springer. (cited on page 89)

200. Sweeney, L., 2002. K-anonymity: A model for protecting privacy. *International Journal of Uncertainty Fuzziness and Knowledge Based Systems*, 10, 5 (2002), 557–570. (cited on pages 29, 98, 101, 167, 190, and 222)

201. Tassa, T. and Cohen, D. J., 2013. Anonymization of centralized and distributed social networks by sequential clustering. *IEEE Transactions on Knowledge and Data Engineering*, 25, 2 (2013), 311–324. (cited on page 73)

202. Taylor, V. F.; Beresford, A. R.; and Martinovic, I., 2017. Intra-library collusion: A potential privacy nightmare on smartphones. *CoRR*, abs/1708.03520 (2017). (cited on page 35)

203. Tinabo, R.; Mtenzi, F.; and O'Shea, B., 2009. Anonymisation vs. pseudonymisation: Which one is most useful for both privacy protection and usefulness of e-healthcare data. In *International Conference for Internet Technology and Secured Transactions*, 1–6. (cited on page 21)

204. Tran, K.-N.; Vatsalan, D.; and Christen, P., 2013. Geco: An online personal data generator and corruptor. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, 2473–2476. ACM, New York, NY, USA. (cited on page 80)

205. Vaidya, J. and Clifton, C., 2002. Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, 639–644. New York, NY, USA. (cited on page 33)

206. Vaidya, J. and Clifton, C., 2005. Secure set intersection cardinality with application to association rule mining. *J. Comput. Secur.*, 13, 4 (2005), 593–622. (cited on page 33)

207. Van Eycken, E.; Haustermans, K.; Buntinx, F.; Ceuppens, A.; WEYLER, J.; Wauters, E.; VAN, O.; et al., 2000. Evaluation of the encryption procedure and record linkage in the Belgian National Cancer Registry. *Archives of public health*, 58, 6 (2000), 281–294. (cited on page 53)

208. Vatsalan, D. and Christen, P., 2012. An iterative two-party protocol for scalable privacy-preserving record linkage. In *AusDM, CRPIT 134*. Sydney, Australia. (cited on pages 26, 27, 29, 56, 68, 69, and 101)

209. Vatsalan, D. and Christen, P., 2013. Sorted nearest neighborhood clustering for efficient private blocking. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 341–352. Springer. (cited on pages 24, 25, 44, 46, and 47)

210. Vatsalan, D. and Christen, P., 2014. Scalable privacy-preserving record linkage for multiple databases. In *ACM CIKM*, 1795–1798. Shanghai. (cited on pages 23, 26, 57, 58, 115, 125, 136, and 160)

211. Vatsalan, D. and Christen, P., 2016. Privacy-preserving matching of similar patients. *Journal of Biomedical Informatics*, 59 (2016), 285–298. (cited on pages 29 and 221)

212. Vatsalan, D.; Christen, P.; O'Keefe, C. M.; and Verykios, V. S., 2014. An evaluation framework for privacy-preserving record linkage. *JPC*, 6, 1 (2014), 35–75. (cited on pages 2, 22, 23, 24, 26, 29, 33, 34, 39, 79, 92, 101, 119, 124, 143, 148, 167, 190, 206, and 215)

213. Vatsalan, D.; Christen, P.; and Rahm, E., 2016. Scalable privacy-preserving linking of multiple databases using counting bloom filters. In *IEEE ICDMW*. (cited on pages 30, 31, 58, 71, 72, 79, and 119)

214. Vatsalan, D.; Christen, P.; and Verykios, V., 2011. An efficient two-party protocol for approximate matching in private record linkage. In *AusDM, CRPIT 121*. Ballarat, Australia. (cited on pages 23, 27, 29, 30, and 56)

215. Vatsalan, D.; Christen, P.; and Verykios, V., 2013. Efficient two-party private blocking based on sorted nearest neighborhood clustering. In *ACM CIKM*, 1949–1958. San Francisco. (cited on pages 26, 27, 29, 41, 50, 62, 112, 123, 124, and 184)

216. Vatsalan, D.; Christen, P.; and Verykios, V. S., 2013. A taxonomy of privacy-preserving record linkage techniques. *JIS*, 38, 6 (2013), 946–969. (cited on pages 2, 4, 7, 21, 22, 24, 39, 41, 46, 52, 57, 61, 79, 90, 91, 100, 114, 148, 160, 163, 167, 182, 191, and 221)

217. Vatsalan, D.; Sehili, Z.; Christen, P.; and Rahm, E., 2017. *Privacy-Preserving Record Linkage for Big Data: Current Approaches and Research Challenges*, 851–895. Springer International Publishing. (cited on pages 1, 2, 20, and 22)

218. Verykios, V.; Karakasidis, A.; and Mitrogiannis, V., 2009. Privacy preserving record linkage approaches. *Int. J. of Data Mining, Modelling and Management*, 1, 2 (2009), 206–221. (cited on pages 2, 21, 22, and 23)

219. Verykios, V. S. and Christen, P., 2013. Privacy-preserving record linkage. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 3, 5 (2013), 321–332. (cited on pages 3, 12, 22, 24, and 35)

220. Vittorini, P.; Angelone, A. M.; Cofini, V.; Fabiani, L.; Mattei, A.; and Necozione, S., 2017. A case study on the integration of heterogeneous data sources in public health. In *International Conference on Bioinformatics and Biomedical Engineering*, 411–423. Springer. (cited on page 3)

221. Wang, G.; Chen, H.; and Atabakhsh, H., 2004. Automatically detecting deceptive criminal identities. *Communications of the ACM*, 47, 3 (2004), 70–76. (cited on pages 7 and 20)

222. Wang, Q.; Vatsalan, D.; and Christen, P., 2015. Efficient interactive training selection for large-scale entity resolution. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 562–573. Springer. (cited on page 23)

223. Weber, S.; Lowe, H.; Das, A.; and Ferris, T., 2012. A simple heuristic for blindfolded record linkage. *Journal of the American Medical Informatics Association*, (2012), 157–161. (cited on page 54)

224. Wen, Z. and Dong, C., 2014. Efficient protocols for private record linkage. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 1688–1694. ACM, New York, NY, USA. (cited on pages 21, 23, 25, 29, 31, and 33)

225. Whang, S. E.; Menestrina, D.; Koutrika, G.; Theobald, M.; and Garcia-Molina, H., 2009. Entity resolution with iterative blocking. In *SIGMOD*. (cited on page 52)

226. Winkler, W. E., 1990. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In *Proceedings of the Section on Survey Research*, 354–359. (cited on pages 19 and 23)

227. Winkler, W. E., 2000. Frequency-based matching in fellegi-sunter model of record linkage. *Bureau of the Census Statistical Research Division*, 14 (2000). (cited on pages 19 and 23)

228. Winkler, W. E., 2006. Overview of record linkage and current research directions. Technical Report RR2006/02, US Bureau of the Census. (cited on pages 9 and 20)

229. Witten, I. H.; Moffat, A.; and Bell, T. C., 1999. *Managing Gigabytes (2nd Ed.): Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. (cited on pages 36, 37, 55, 84, and 141)

230. Xiao, C.; Wang, W.; Lin, X.; and Yu, J. X., 2008. Efficient similarity joins for near duplicate detection. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, 131–140. ACM, New York, NY, USA. (cited on page 51)

231. Yakout, M.; Atallah, M.; and Elmagarmid, A., 2009. Efficient private record linkage. In *IEEE International Conference on Data Engineering*, 1283–1286. (cited on pages 21, 26, 28, 29, and 55)

232. Yang, B.; Sato, I.; and Nakagawa, H., 2012. Privacy-preserving em algorithm for clustering on social network. In *PAKDD*. (cited on page 71)

233. Yao, A., 1986. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, 162–167. IEEE. (cited on pages 27 and 30)