Relational Data Curation by Deduplication, Anonymization, and Diversification

RELATIONAL DATA CURATION BY DEDUPLICATION, ANONYMIZATION, AND DIVERSIFICATION

BY

YU HUANG, Ph.D

Ph.D (2020) McMaster University

(Computer Science) Hamilton, Ontario, Canada

TITLE:	Relational Data Curation by Deduplication, Anonymization, and Diversification
AUTHOR:	Yu Huang Doctoral, (Computer Science)
SUPERVISOR:	Professor Fei Chiang
NUMBER OF PAGES:	xiii, 165

Dedicated to my family and friends.

Abstract

Enterprises acquire large amounts of data from a variety of sources with the goal of extracting valuable insights and enabling informed analysis. Unfortunately, organizations continue to be hindered by poor data quality as they wrangle with their data to extract value since most real datasets are rarely error-free. Poor data quality is a pervasive problem that spans across all industries causing unreliable data analysis, and costing billions of dollars [1, 2]. The large body of datasets, the pace of data acquisition, and the heterogeneity of data sources pose challenges towards achieving high quality data. These challenges are further exacerbated with data privacy and data diversity requirements. In this thesis, we study and propose solutions to address data duplication, managing the trade-off between data cleaning and data privacy, and computing diverse data instances.

In the first part of this thesis, we address the data duplication problem. We propose a duplication detection framework, which combines word-embeddings with constraints among attributes to improve the accuracy of deduplication. We propose a set of constraint-based statistical features to capture the semantic relationship among attributes. We showed that our techniques achieve comparative accuracy on real datasets. In the second part of this thesis, we study the problem of data privacy and data cleaning, and we present a Privacy-Aware data Cleaning-As-a-Service (PACAS)

framework to protect privacy during the cleaning process. Our evaluation shows that PACAS safeguards semantically related sensitive values, and provides lower repair errors compared to existing privacy-aware cleaning techniques. In the third part of this thesis, we study the problem of finding a diverse anonymized data instance where diversity is measured via a set of diversity constraints, and propose an algorithm to seek a k-anonymous relation with value suppression as well as satisfying given diversity constraints. We conduct extensive experiments using real and synthetic data showing the effectiveness of our techniques, and improvement over existing baselines.

Acknowledgements

I would first like to thank my supervisor, Professor Fei Chiang, for her invaluable guidance and patience in helping me with my research all these years. Her insightful advice helps me sharpen my critical thinking and make me a better researcher. Without her support and wise advice, I would not have been able to accomplish this thesis.

Besides my supervisor, I would like to express my appreciation to my thesis committee members, Professor Frantisek Franek, Professor Wenbo He and Professor Reza Samavi, for their insightful comments and encouragement on this thesis. Your feedback widen my research from various perspectives. I would like to thank Professor Farhana Zulkernine, for serving as my external reviewer and for reading my thesis and providing valuable feedback.

I would like to thank all group members in our Data Science Lab: Zheng Zheng, Morteza Langouri, Dhruv Gairola, Qu Zhi, Levin Noronha, Harika Gorla and many others, for the helpful discussions in my research. Thank you to Mostafa Milani, I learned a lot from the collaboration and discussion with you.

Finally, to my wife, Yu Sun, thank you for your understanding and support over the years. To my parents, thank you for everything.

Contents

Abstract **iv**

Acknowledgements **vi**

1 Introduction **1**

1.1 Data Deduplication 4

1.2 Data Privacy 6

1.3 Data Diversity 9

1.4 Contributions 11

2 Preliminaries **13**

2.1 Relations and Functional Dependencies 13

2.2 k-Anonymity 15

3 Semantic-Aware Deduplication **17**

3.1 Introduction 17

3.2 Related Work 21

3.2.1 Entity Matching Frameworks 21

3.2.2 Semantic-based Entity Resolution 22

3.2.3 Machine Learning Approaches 23
3.3 Preliminaries 24
3.3.1 Metric Functional Dependencies 25
3.3.2 Word-Embeddings 26
3.4 Framework Overview 27
3.4.1 Problem Statement 27
3.4.2 Solution Overview 28
3.5 Non-constraint Attributes Feature Generation 33
3.5.1 Edit Distance 34
3.5.2 q-grams Similarity 35
3.5.3 Term Differentiation 36
3.5.4 Weighted Frequency Similarity 37
3.6 Constraint Attributes Feature Generation 38
3.6.1 Constraint Properties 39
3.6.2 The Number of Satisfied Constraints 39
3.6.3 Matching Constraint Attributes 40
3.6.4 Proportion of Exact Matching Attributes 40
3.6.5 MFD Tolerance Parameter 41
3.7 Duplication Metric 42
3.7.1 Duplication of an Attribute Value 42
3.7.2 Duplication across an Attribute Domain 43
3.8 Limitations 44
3.9 Evaluation 48
3.9.1 Experimental Setup 49

3.9.2 Comparative Accuracy: Term Differentiation and Similarity Functions . . . 52
3.9.3 Weighted Term Frequency Efficiency . . . 54
3.9.4 Utility of Constraint Features . . . 55
3.9.5 Varying the Size of Training Data. . . . 55
3.9.6 Statistical Feature Analysis . . . 56
3.9.7 Sensitivity to Δ . . . 58
3.9.8 Accuracy of Duplication Scores . . . 59
3.9.9 Comparative Baseline Evaluation . . . 60
3.9.10 Runtime Evaluation . . . 62
3.10 Conclusion . . . 63

4 Privacy-Preserving Data Cleaning 65

4.1 Introduction . . . 65
4.2 Related Work . . . 74
4.3 Preliminaries . . . 76
4.3.1 Matching Dependencies . . . 77
4.3.2 (X,Y)-Anonymity . . . 77
4.3.3 Generalization . . . 79
4.3.4 Data Pricing . . . 80
4.4 Generalized Relations . . . 81
4.4.1 Measuring Semantic Distance . . . 82
4.4.2 Consistency in Generalized Relations . . . 84
4.5 PACAS Overview . . . 85
4.5.1 Problem Statement . . . 85

4.5.2 Solution Overview 86
4.6 Limiting Disclosure of Sensitive Data 87
4.6.1 Record Matching and Query Generation 88
4.6.2 Enforcing Privacy via Data Pricing 89
4.6.3 Query Answering 95
4.7 Data Cleaning with Generalized Values 95
4.7.1 Overview 96
4.7.2 Generating Equivalence Classes 99
4.7.3 Selecting Equivalence Classes for Repair 100
4.7.4 Data Request Generation 101
4.7.5 Purchase Data and Repair 102
4.7.6 Complexity Analysis 103
4.8 Experiments 103
4.8.1 Experimental Setup 104
4.8.2 Generalized Values 108
4.8.3 *SafePrice* Efficiency and Effectiveness 110
4.8.4 *SafePrice* Parameter Sensitivity 111
4.9 Conclusions 114

5 Diversifying Anonymized Data with Diversity Constraints 116

5.1 Introduction 116
5.2 Related Work 122
5.3 Preliminaries 124
5.3.1 Diversity Constraints 124
5.4 The DIVA Algorithm 125

5.4.1 Diverse Clustering 127
5.5 Selection Strategies 133
5.6 Experiments 134
5.6.1 Experimental Setup 135
5.6.2 Metrics and Parameters 139
5.6.3 Accuracy 140
5.6.4 Performance 143
5.6.5 Overhead of Diversity Constraints 143
5.7 Conclusion 145

6 Conclusion and Future Work **146**

6.1 Data Deduplication 146
6.2 Data Privacy and Data Cleaning 147
6.3 Data Diversity 148

List of Figures

1.1 Three properties of data quality 2

1.2 Integrated data quality pipeline 2

3.1 Deduplication Framework Overview. 27

3.2 Similarity graph for an attribute value. 42

3.3 Similarity graph for an attribute domain. 44

3.4 Comparative precision. 53

3.5 Comparative recall. 53

3.6 Comparative precision. 53

3.7 Comparative recall. 53

3.8 Term diff: precision. 54

3.9 Term diff: recall. 54

3.10 Constraints: precision. 54

3.11 Constraint: recall. 54

3.12 Precision (DBLP-Scholar). 56

3.13 Recall (DBLP-Scholar). 56

3.14 Precision (Amazon-Google) 57

3.15 Recall (Amazon-Google) 57

3.16 Feature weights 58

3.17 Δ Sensitivity . . . 58
3.18 Dup. scores (Restaurant) . . . 58
3.19 Dup. scores (Abt-Buy) . . . 59
3.20 Dup. scores (DBLP-ACM) . . . 59
3.21 Comparative Precision . . . 60
3.22 Comparative Recall . . . 60
4.1 (a) $\mathsf{DGH}^{\mathsf{med}}$ and (b) $\mathsf{VGH}^{\mathsf{med}}$. . . . 66
4.2 (a) $\mathsf{DGH}^{\mathsf{age}}$ and (b) $\mathsf{VGH}^{\mathsf{age}}$. . . . 66
4.3 Framework overview. . . . 86
4.4 Possible relations $\mathcal{I}$, conflict set $\mathcal{C}_Q$ and admissible relations $\mathcal{I}_Q$ for query Q. . . . 91
4.5 Evaluation on gen. values . . . 109
4.6 Repair error vs. $|\mathcal{S}|$. . . . 110
4.7 *SafePrice* Parameter Sensitivity . . . 111
4.8 Comparative repair error. . . . 113
4.9 Comparative runtime. . . . 113
5.1 Diverse clustering as graph coloring. . . . 132
5.2 Varying Σ, k, *cf.* . . . 140
5.3 *DIVA* effectiveness and efficiency. . . . 142
5.4 Comparative evaluation. . . . 144

Chapter 1

Introduction

Organizations have found it increasingly difficult to extract insights from their data due to increasing data volume and heterogeneity. Many data processing tasks assume the data conforms to standard formats and data types, which is rare in real data. Data acquired from the imprecision of extractors may contain missing values, data integrated from heterogeneous sources may introduce duplicate tuples, and human typos in data entry may violate declared constraints. Data quality is a pervasive problem that spans across all industries. Recent studies report that the financial impact of poor data quality is approximated at $600B per year on US businesses [3]. A large number of companies indicating that managing data quality remains their number one issue. As data grows in complexity and size, the data quality issues become essential to gain accurate and useful information. However, dealing with these data quality issues is not a trivial problem as data errors could come from various sources and in different formats.

In this thesis, we focus on three aspects of data quality as Figure 1.1 shows: (i) data deduplication, (ii) privacy-aware data cleaning; (iii) data diversity.

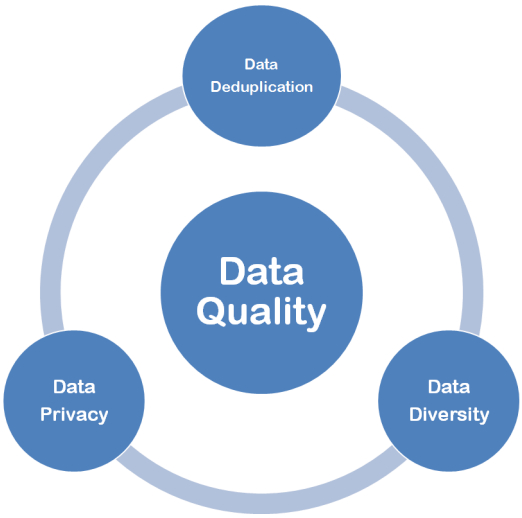


Figure 1.1: Three properties of data quality

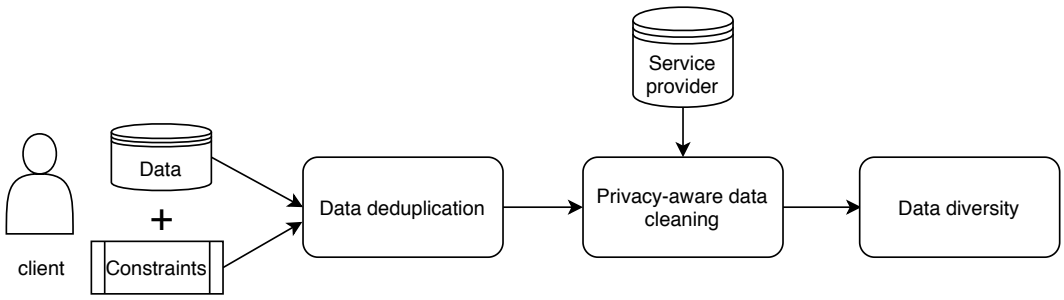


Figure 1.2: Integrated data quality pipeline

These three properties of data quality are essential for data analysis and data mining. Since data could come from multiple sources across different platforms or organizations, data deduplication is an essential step when we integrate data from different sources before analysis. Besides deduplication, we also need to ensure that data is error-free through data cleaning and protect data privacy during the process. To protect data privacy, a widely use privacy model is k-anonymity, which uses generalization and suppression to protect sensitive values. The data anonymization technique may lead to biased data by masking or excluding minority groups, so we have to consider data diversity during the anonymization process as well.

In this thesis, we present techniques to resolve three properties of data quality. Figure 1.2 shows an overview of our data pipeline. A relational dataset with predefined integrity constraints provided by the client is used as an input of the data pipeline. The integrity constraints are a set of rules that describe the semantic relationship among attributes of the relational dataset, which are formally defined in Section 2. The dataset may contain errors (such as duplicates, inconsistent values), which need to be fixed. The first component of this pipeline is the data deduplication module, which uses the given constraints to identify and remove duplicates. The output of the data deduplication component is a data instance without duplicates. Since deduplication cannot fix inconsistent values in the data, we propose the second component of the pipeline, the privacy-aware data cleaning component. This component sends query requests to a service provider (who owns accurate and correct data) asking for correct data to fix the inconsistent values. However, the service provider is not publicly accessible because of privacy restrictions. The communication between the service provider and the client should use a privacy-preserving protocol,

and we propose a privacy-preserving data cleaning technique to handle it. Although the privacy-aware data cleaning component can deal with inconsistent errors, it does not maintain the diversity property of the data, which causes underlying bias in subsequent data analysis. Hence, we design the data diversity component as the third component of our pipeline to ensure the integrated data is unbiased and diverse.

Our framework can be extended as an end-to-end data pipeline to deal with data quality issues as well. The input of the pipeline is a dataset with predefined constraints (such as integrity constraints and diversity constraints). The integrity constraints will be used in the data deduplication component and privacy-aware data cleaning component, and the diversity constraints will be used in the data diversity component. The output of data deduplication, a data instance without duplicates, will be used as input for the data cleaning component. The output of the data cleaning component is a consistent data instance. This consistent data instance with the predefined diversity constraints can be used as the input for the data diversity component to ensure data diversity property.

1.1 Data Deduplication

id	**Symbol**	**Name**	**IPOyear**	**Sector**	**Industry**
t_1	MMM	3M Corporation	2003	Technology	Electrical Products
t_2	DDD	3D Corporation	2013	Technology	Electrical Products
t_3	ABB	ABB Corporation	1992	Consumer Durables	Industrial, Machinery
t_4	ABM	ABM Industries Ltd	2001	Consumer Durables	Industrial, Machinery
t_5	AFL	Aflac Incorporated	2011	Finance	Health Insurance

Table 1.1: Duplicate tuples.

Data duplication occurs when a real-world entity has two or more different representations within or across databases. Ideally, in an error-free system with perfectly clean data, each tuple in a database has a unique identifier. Unfortunately, in most practical cases, this does not occur, and the data often lacks a unique, global identifier. This especially occurs in data integration when data is integrated from multiple sources across different departments or organizations. In such cases, it is inevitable to introduce duplicates due to differences in tuple format, standardizations, schema and typos. Hence, identifying duplicates in the data is a critical step towards ensuring reliable and trusted data.

Data deduplication is a costly and tedious task that involves identifying duplicate tuples in a dataset. High duplication rates lead to poor data quality, where data ambiguity occurs as to whether two tuples refer to the same entity. The process of tuple deduplication, also known as tuple linkage, is the process of reconciling tuples that refer to the same entity. However, identifying duplicates is not a trivial problem. Different tuple semantics, syntax, even the frequency of terms may affect the accuracy of identified duplicates. One solution is to calculate the string similarity score between two tuples and compare the score to a given threshold to identify duplicates [4]. String similarity metrics consider all values as strings, and typically measure the number of edits needed to transform one string to the other. This process is far from perfect. For example, in Table 1.1 if we compared the company name of 3D Corporation and 3M Corporation in t_1 and t_2 through string similarity, we intuitively identify these two are duplicates because there is only one character difference. However, these two tuples are distinct companies by focusing on the differentiating terms 3D and 3M, rather than the terms Company or Corporation, which also occur frequently in other

tuples.

To address this challenge, we propose a deduplication framework that differentiates terms during the similarity matching step, and is agnostic to the ordering of values within a tuple. We propose a set of constraint features to capture the semantic relationship among attributes. Our framework combines constraint features with non-constraint features to detect duplicates. We propose a duplication metric that quantifies the level of duplication for an attribute value, and within an attribute. This metric can be used by analysts to understand the distribution and similarity of values during the data cleaning process.

1.2 Data Privacy

Given the proliferation of sensitive, confidential user information, data privacy concerns have largely remained unexplored in data cleaning techniques. As increasing amounts of personal and sensitive information are collected online by social media sites and by organizations, there is growing concern about how to perform automated data cleaning tasks while ensuring sensitive and confidential information remains protected. Data privacy has become increasingly important, and new techniques for improving data quality while ensuring minimal data disclosure are strongly needed. Developing automated techniques to provide data privacy guarantees during data cleaning is challenging, as these two tasks have competing goals. In data cleaning, the objective is to gain as much knowledge about the data as possible to resolve data errors correctly. In data privacy, the goal is to conceal as much of the data as possible.

In declarative data cleaning, a set of data updates (repairs) are generated based on transforming the dirty data values to match clean and consistent portions of the

ID	GEN	ETH	PRV	CTY	DIAG	MED
t_1	Female	Caucasian	AB	Calgary	Osteoarthritis	**Ibuprofen**
t_2	Female	Caucasian	AB	Calgary	Osteoarthritis	**Addaprin**
t_3	Male	Caucasian	AB	Calgary	Osteoarthritis	Naproxen
t_4	Male	Caucasian	MB	Winnipeg	Ulcer	Tylenol
t_5	Male	African	MB	Winnipeg	Osteoarthritis	Naproxen
t_6	Male	African	MB	Winnipeg	Migraine	Dolex

Table 1.2: Client's table with errors

data. These repairs are generated from an external source, which is assumed to be trusted and clean. Generating a set of repair recommendations requires accessing and reporting both the erroneous and clean data values. Current approaches assume that all data values are equally informative [5], and are fully accessible. In reality, this is often not the case. Organizations invest significant efforts to protect their proprietary data, where a set of attribute values are highly confidential, providing higher information content than the remaining attribute values. For example, a patient's medical tuple describing her diagnosed illness and prescribed medications is more identifying and informative than his general symptoms, residential city, and gender. These sensitive values containing confidential information that are not publicly accessible, but are used as a trusted source for data cleaning. These sensitive values often have restricted access, either providing limited viewing to a subset of the data, or obscuring the true value(s) via generalization (e.g., revealing a person's age group rather than her actual age). Unfortunately, existing data cleaning approaches do not consider these restrictions, thereby leading to inadvertent disclosure of private data

as part of a repair.

There are two agents in the data cleaning process. One agent is the client with data errors. Another agent is the service provider, who owns the clean data. We assume both client and service provider have the same schema, and the dataset owned by the client is a subset of the dataset owned by the service provider. If the client wishes to clean his data with the service provider's help, he has to send query requests to the service provider asking for correct data. A set of data repairs will be generated from the service provider to fix the error in the client's dataset. For example, Table 1.2 and Table 1.3 are the client's table and service the provider's table, respectively. In Table 1.2 and Table 1.3 the schema consists of patient gender (GEN), ethnicity (ETH), province (PRV), city (CTY), diagnosed illness(DIAG), and prescribed medication (MED). Suppose there is an integrity constraint [GEN,DIAG] $\rightarrow$ [MED] defined over Table 1.2, which states states a person's gender and diagnosed condition determine a prescribed medication. It indicates that the values in attributes GEN and DIAG uniquely determine the value in MED. According to this constraint, tuples t_1 and t_2 violate the given constraint, and error cells are highlighted in Table 1.2. To fix the errors in Table 1.2, a set of data repairs need to be generated from Table 1.3. However, these repairs may reveal sensitive values about diagnosed illness and medication in Table 1.3. The challenge we address is: "How can we ensure the communication between client and service provider being under a privacy-preserving protocol during the data cleaning process?"

In this thesis, we present a privacy-aware, data cleaning framework that aims to resolve data inconsistencies while minimizing the amount of information disclosed. This framework aims to balance the tradeoff between data privacy and data utility.

ID	GEN	ETH	PRV	CTY	DIAG	MED
m_1	Female	Caucasian	AB	Calgary	Osteoarthritis	Ibuprofen
m_2	Female	Caucasian	AB	Calgary	Tendinitis	Addaprin
m_3	Male	Caucasian	AB	Calgary	Migraine	Naproxen
m_4	Male	Caucasian	MB	Winnipeg	Ulcer	Tylenol
m_5	Male	African	MB	Winnipeg	Osteoarthritis	Ibuprofen
m_6	Male	African	MB	Winnipeg	Migraine	Dolex

Table 1.3: Service provider's table.

1.3 Data Diversity

ID	GEN	ETH	PRV	CTY	DIAG	MED
t_1	*	Caucasian	AB	Calgary	Osteoarthritis	Ibuprofen
t_2	*	Caucasian	AB	Calgary	Tendinitis	Addaprin
t_3	*	Caucasian	AB	Calgary	Migraine	Naproxen
t_4	Male	*	MB	Winnipeg	Ulcer	Tylenol
t_5	Male	*	MB	Winnipeg	Osteoarthritis	Ibuprofen
t_6	Male	*	MB	Winnipeg	Migraine	Dolex

Table 1.4: Table with anonymization ($k = 3$).

To protect sensitive values, a widely used privacy model is k-anonymity, which intends to break the linkage between identifiers and sensitive attributes [6]. In k-anonymity, quasi-identifier(QI) attributes are the attributes that can be used to identify an individual. According to k-anonymity, for each tuple in a k-anonymous table there are at least k tuples with the same quasi attributes. This means the individual

represented by the tuple can not be identified, because the attacker cannot distinguish it from $k-1$ other individuals. The k-anonymity model generates a k-anonymous relation through an anonymization process, such as generalization and suppression [6]. Real data is sensitive and requires anonymization, but also contains characteristic details from a variety of individuals. This diversity is desirable in many applications ranging from Web search to drug and product development. To avoid biased decision making, incorporating diversity into computational models is essential to prevent and minimize discrimination against disadvantaged and minority groups. Unfortunately, data anonymization techniques have largely ignored diversity in its published result, which causes underlying bias in subsequent data analysis.

For example, Table 1.3 presents patients' medical tuples with sensitive attribute diagnosed illness (DIAG). After applying k-anonymity to Table 1.3 with $k = 3$, we obtain anonymized Table 1.4, where each group has at least three tuples with the same quasi attributes to break the linkage between QI attributes and sensitive attributes.

The anonymization progress masks some attribute values, which may exclude minority groups and cause biased decision making in data analysis. For example, in Table 1.4 the values of "Female" under attribute GEN has been replaced with "*". To avoid bias and ensure diversity in the dataset, existing work has proposed a declarative method in the form of diversity constraint [7]. Data diversity constraint defines the expected frequencies for the attribute values in the data, which can ensure these values not being masked and excluded during anonymization. Diversity constraints provide a declarative definition of the minimum and maximum frequency bounds that specific attribute domain values should appear in the table, which can guarantee the minority group (such as "Female") will not be totally excluded. An example

of a diversity constraint $\sigma_1 = (ETH[Female], 2, 4)$ requires Table 1.4 should contain a minimum of two "Female" values and no more than four on the attribute ETH. The challenge is to generate an anonymized table that satisfies the requirements of k-anonymity and data diversity at the same time.

In this thesis, we couple diversity with k-anonymity. We study the problem of finding a diverse anonymized data instance where diversity is measured via a set of diversity constraints.

1.4 Contributions

In this thesis, we aim to provide an end-to-end pipeline to address three pressing data quality challenges, namely: data duplication, data privacy and data cleaning, and data diversity. We make the following contributions:

1. We propose a duplication detection framework, which combines word embeddings with constraints among attributes to improve the accuracy of deduplication. We propose a set of constraint-based statistical features to capture the semantic relationships among attributes. We present a duplication metric that quantifies the level of duplication to understand the distribution and similarity of values during the data cleaning process. This work was published on CASCON 2017 [8] and TPDL 2017 [9].

2. To address the data privacy requirements, we propose a new *Data Cleaning-as-a-Service* model that allows a client to interact with a data cleaning provider who hosts curated, and sensitive data. We present *PACAS*: a Privacy-Aware data Cleaning-As-a-Service framework that facilitates communication between

the client and the service provider via a data pricing scheme where clients issue queries, and the service provider returns clean answers for a price while protecting her data. We propose a practical privacy model in such interactive settings called (X,Y,L)-anonymity that extends existing data publishing techniques to consider the semantics in the data while protecting sensitive values. This work was published on WISE 2015 [10], IEEE Big Data 2018 [11] and the Journal of Information Systems [12].

3. To ensure data diversity, we studied the problem of diversifying anonymized data using diversity constraints. We present a diversity-driven anonymization algorithm that anonymizes a given data instance, while ensuring a set of diversity constraints are satisfied. We conduct an extensive evaluation using real datasets demonstrating the effectiveness and efficiency of our algorithm and show the utility of diversity constraints over an existing baseline.

In Chapter 2, we introduce the necessary notations and definitions that are used throughout this thesis. In Chapter 3, we present our deduplication framework and perform extensive experiments showing that our model can detect duplicates with good accuracy. In Chapter 4, we present our privacy-preserving data cleaning framework which provides privacy guarantee during data cleaning. In Chapter 5, we present our diversity-driven anonymization algorithm which guarantees data privacy and data diversity at the same time. Finally, we conclude and propose future work in Chapter 6.

Chapter 2

Preliminaries

In this chapter, we present the necessary notations and definitions that are used throughout the thesis. We summarize the notations and definitions in Table 2.1. Notation and definitions relevant to a unique chapter are introduced in the specific chapter.

2.1 Relations and Functional Dependencies

A *relation* (table) R is a finite set of n-ary tuples $\{t_1, t_2..., \}$. A *database* D is a finite set of relations. The relational attributes are denoted as $A_1, A_2, ...$, and sets of attributes are denoted as X, Y.

Definition 2.1.1. A *functional dependency* (FD) φ over a relation R is denoted by $\varphi : X \rightarrow Y$, where X and Y set of attributes. For every pair of tuples t and t', if $t[X] = t'[X]$ implies $t[Y] = t'[Y]$, we say relation R satisfies φ (denoted as $R \models \varphi$).

The definition of functional dependency states that if two tuples agree on the X attribute values, then they musl also agree on the Y attribute values [13]. For

example, FD φ: CITY $\rightarrow$ PROVINCE holds in Table 2.2. It indicates that given any two tuples in Table 2.2, if they have identical values on the CITY attribute $t_1[CITY] = t_2[CITY]$, then they should have identical values on the PROVINCE attribute $t_1[PROVINCE] = t_2[PROVINCE]$.

Symbol	Description
R	Relation
D	Database instance
$A_1, A_2...$	Relational attributes
X, Y	Sets of relational attributes
t	A tuple of R
φ	Functional dependency (FD)
Σ	A set of constraints

Table 2.1: Summary of notation and symbols.

We focus on FDs because they are frequently used to enforce data consistency and are essential in helping to maintain and improve the data quality [14, 15, 16]. Many database management systems follow the relational model and use FDs to control the quality of the data which is stored in the tables [15, 16, 17, 18, 19]. For example, declarative data cleaning approaches have focused on achieving the consistency of the given FDs to ensure the data quality, and data values that violate the FDs are identifed as errors that need to be repaired [15, 16, 20, 21]. There are a wide range of data cleaning algorithms (based on user/expert feedback [22, 23, 24], master database [14, 15], knowledge bases or crowdsourcing [25], probabilitic inference [18, 19]) leverage FDs to identify and fix errors.

ID	GEN	AGE	CITY	PROVINCE
m_1	male	51	Toronto	ON
m_2	female	45	Toronto	ON
m_3	female	32	Vancouver	BC
m_4	female	67	Vancouver	BC
m_5	male	61	Edmonton	AB
m_6	male	79	Edmonton	AB

Table 2.2: Relational table

2.2 k-Anonymity

A classic privacy-preserving data publishing model is k-anonymity, which prevents re-identification of an individual in an anonymized data set [6, 26]. Attributes in a relation are either *identifiers* such as SSN that uniquely identify an individual, *quasi-identifier* (QI) attributes such as ethnicity, address, age that together can identify an individual, or *sensitive* attributes that contain personal information.

Definition 2.2.1 (QI-group and k-anonymity)**.** A relation R is k-anonymous if every QI-group has at least k tuples. A QI-group is a set of tuples with the same values in the QI attributes.

k-anonymity is known to be prone to attribute linkage attacks where an adversary can infer sensitive values given QI values. As an example, Table 2.3 is the original table and Table 2.4 is the k-anonymous table with $k = 3$. Table 2.4 has two QI-groups, $\{g_1, g_2, g_3\}$ and $\{g_4, g_5, g_6\}$.

In this chapter, we introduce necessary notations about relation, attributes and

ID	GEN	AGE	ZIP	DIAG	MED
m_1	male	51	P0T2T0	osteoarthritis	ibuprofen
m_2	female	45	P2Y9L8	tendinitis	addaprin
m_3	female	32	P8R2S8	migraine	naproxen
m_4	female	67	V8D1S3	ulcer	tylenol
m_5	male	61	V1A4G1	migraine	dolex
m_6	male	79	V5H1K9	osteoarthritis	ibuprofen

Table 2.3: Original medical records table

ID	GEN	AGE	ZIP	DIAG	MED
g_1	*	[31,60]	P*	osteoarthritis	ibuprofen
g_2	*	[31,60]	P*	tendinitis	addaprin
g_3	*	[31,60]	P*	migraine	naproxen
g_4	*	[61,90]	V*	ulcer	tylenol
g_5	*	[61,90]	V*	migraine	dolex
g_6	*	[61,90]	V*	osteoarthritis	ibuprofen

Table 2.4: k-anonymous medical records table

functional dependency. We also provide the formal definition of functional dependency and k-anonymity. In the next chapter, we will discuss data deduplication, which is the first component of our data pipeline.

Chapter 3

Semantic-Aware Deduplication

3.1 Introduction

Organizations rely on the results of analyzing data to make business decisions and develop strategies. The validity of data analysis depends on the quality of the data. Organizations typically assume data is consistent, accurate and conforms to standard formats and data types. However, as data is gathered from various sources, it may contain missing values, duplicates, inconsistencies, mixed formats and human typos [2]. Data duplication is a critical data quality issue. It occurs when an entity has two or more different representations within or across databases. Ideally, in an error-free system with high-quality data, each tuple in a database should include a unique identifier. Unfortunately, in most practical cases, data often lacks a unique, global identifier because of their heterogeneous origins. In such cases, duplicate detection is an essential step towards ensuring reliable high-quality data for a broad range of significant data analysis tasks, including data fusion and data profiling. The process of detecting duplicates, which is also known as *entity resolution* (ER) [27], *record*

linkage [28], or *entity matching* (EM) [29] is to reconcile tuples that refer to the same entity [30]. In this chapter, we use the term entity resolution (ER) to describe this process.

Identifying duplicates is not a trivial task, especially when the tuples contain similar semantic terms (such as *Mobile* vs *Phone* in Table 3.5). Existing approaches rely on string similarity of attribute values to identify duplicates [4, 31]. String similarity is commonly used to identify duplicates. It works well when the similarity score of values in two tuples is high. However, the limitation of string similarity is that it cannot detect duplicates when two tuples have low string similarity but have high semantic similarity. By semantic similarity, we refer to how close two terms are to represent the same entity, independent of their syntactic (string) representation. For example, *Mobile* vs *Phone* are two terms that are semantically similar but their string similarity score is low. Existing lexical matching approaches such as edit distance only consider the number of different characters to compute the similarity, and cannot capture similar words on the semantic level [31]. Incorporating a semantic similarity measure In this chapter, we use the word-embedding techniques to capture the semantic similarity between terms.

tuple	Product	Model	Price	Category
t_1	Apple iPhone ios 8 32GB Cell Phone	IPH1706	1307	Mobile
t_2	Samsung 1080P LED 40 high-definition smart FPTV	SM900L	960	Video
t_3	Apple iPhone ios 8th generation 32GB Mobile	IPH1706	1367	Phone
t_4	Samsung 1080P LED 40 HD Flat-panel Television	SM900L	943	Video

Table 3.5: Product Dataset.

Term Frequency. Existing deduplication techniques assume that the relative importance of the terms is equal when calculating the string similarities and ignore to consider the relative weights of each term within attribute values [4, 30, 31]. For example, given two attribute values *ABB company* and *IBM company*, if we treat each term with equal weight, the similarity score of these two values will be 0.8 (since only two characters out of ten are different). These two values will be considered as duplicates. However, if we consider the weight of each term, we could intuitively identify these two are different companies by putting more weight on the differentiating terms *ABB* and *IBM*, rather than the common term *company*. In this chapter, we consider the relative importance of each term within attribute values and propose a weighted mechanism based on term frequency when we calculate the similarity score.

Data integrity constraints provide useful information of expected attribute relationships to identify duplicates. If such constraints are available, they provide additional context beyond traditional attribute similarity scores. For example, an extension of functional dependencies (FDs) are *Metric Functional Dependencies (MFDs)*, which allow small differences to occur in the consequent values of the constraint within a tolerance parameter Δ [32]. For an MFD $\omega : X \xrightarrow{\Delta} Y$, if two tuples $t[X] = t'[X]$ then $d(t[Y] - t'[Y]) \leq \Delta$, for a metric distance function d over numeric values in Y. For example, in Table 3.5 assume we have a MFD that states $[Model, Category] \xrightarrow{50} [Price]$, which indicates that the price of a product should be within a certain variance (50) given the model number and category. By using an MFD, we know that if two tuples have the same value on attribute Model and Category, then minor differences in Price are acceptable. The metric parameter Δ is defined as part of ω and normally set by users according to application requirements and allowed tolerance levels. A

relation R may have many dependencies defined, including FDs (denoted as φ) and MFDs (denoted as ω). We call this set of constraints Σ, and say that $R \models \Sigma$ if for all $\varphi, \omega \in \Sigma$, $R \models \varphi$ and $R \models \omega$. We assume that Σ is given and defined by data architects according to the domain semantics.

In this chapter, we develop a deduplication framework that leverages integrity constraints, and consider entity resolution as a binary classification problem. We propose a semantic-aware framework to train a classifier to classify unlabeled tuple pairs as duplicates. In this chapter, we consider the commonly used FDs and MFDs, and provide a methodology to extract features from these constraints towards identifying duplicates. This chapter makes the following contributions:

1. We present a unified framework for identifying duplicates that leverages integrity constraints and word-embeddings to improve the accuracy of identifying duplicates.

2. We propose a set of statistical features that can measure the semantic relationships among attributes through FDs and MFDs for learning.

3. We propose a weighting scheme that differentiates terms (in a tuple) based on their frequency of occurrence. This helps to place greater emphasis on terms that are better indicators of duplicates, and discounting commonly occurring terms that do not serve as good differentiators.

4. We present a new duplication score that quantifies the degree of duplication for values in an attribute domain. This metric allows users to measure the duplicate score between attribute values as well as the level of data duplication between columns. This metric can be used by analysts to understand the distribution

and similarity of values during the data cleaning process.

5. We conduct a comprehensive experimental evaluation of our techniques with respect to accuracy and performance. We also show that our duplication scores are intuitive, and capture observed duplication patterns in the data.

This chapter is organized as follows. We introduce related work in Section 3.2, and describe necessary preliminaries in Section 3.3. In Section 3.4, we formalize the problem statement, present a solution overview, and introduce the modules of our framework. In Section 3.5 we discuss how to generate non-constraint attribute features, including similarity functions and term differentiation. In Section 3.6, we describe how constraint-based attribute features are computed. We then define a set of metrics to measure the level duplication in an attribute domain in Section 3.7. In Section 3.8, we discuss the limitations of our model and provide some possible solutions. In Section 3.9, we present our experimental results, and conclude in Section 3.10.

3.2 Related Work

3.2.1 Entity Matching Frameworks

Cohen presents a system named WHIRL which uses tf-idf weights with cosine similarity to calculate string similarity [33]. WHIRL divides a tuple into a set of terms, and for each term, the term frequency (TF) to inverse document frequency (IDF) score is used to represent the term. TF is the number of times that word appears in the string, which is $TF = f_{(w,d)}$. IDF is the inverse document frequency $IDF = |D|/N$,

where N is the number of records in the database D that contain the word w, and $|D|$ is the total number of documents in D. The TF-IDF value is computed as $(TF \cdot IDF) = \log(TF + 1) * \log(IDF)$. The idea behind the TF-IDF score is that if a term appears many times in one tuple (large TF), but relatively few times in the table (large IDF), then TF-IDF measures its relative significance.

The Stanford Entity Resolution Framework (SERF) identifies duplicates via a matching phase followed by a merging phase [34, 35]. In the matching phase, they use a matching function to calculate the similarity scores of attribute values. If the similarity scores are above a given threshold, a boolean function returns true indicating a duplicate has been identified. This process repeats across all attributes, and a pair of tuples are considered as duplicates if the logical AND across the attributes is true. In the merging phase, they use a merging function to merge two tuples into a consolidated tuple representing a unique entity.

Gustavo et al. propose a blocking scheme, which is based on an inverted index structure to cluster entities in blocks [36]. This method can handle duplicate pairs in each block rather than comparing the entire data instance. Wang et al. present a blocking framework based on locality-sensitive hashing (LSH) function to reduce the number of tuple comparisons [37].

3.2.2 Semantic-based Entity Resolution

Salima et al. propose an ontology-based approach to alleviate the ambiguity of entities from various sources [38]. They use the ontology information of the dataset to manually express the semantic connections between concepts. They assume the same

entities belong to the same family of concepts in the ontology and use the given ontology tree to detect all possible connections between entities. However, their approach heavily relies on the pre-defined ontology hierarchies, which are not always easy to obtain. Building these ontology hierarchies requires manual and costly effort from domain experts as well.

3.2.3 Machine Learning Approaches

The proliferation of machine learning techniques has naturally been applied to duplication detection problems. Muhammad et al. propose a learning-based framework, where they obtain distributed representations of words, and build a classifier to identify duplicates [39]. To capture the semantic meaning of the words in the entity, they use distributed representations, such as word2vec, to map an entity into a vector, then calculate the similarity among the vectors to check for similar entities.

Magellan is an end-to-end deduplication framework [40]. It treats the deduplication as a binary classification task, by converting tuple attributes to features to train the classifier [40]. Although Magellan performs well on clean, structured records, it is not robust enough on tuples with many synonyms. The extracted features use attribute string similarity which fails to identify the closeness of synonyms. Another limitation of Magellan is that it does not consider the relationships among attribute, and it evaluates each attribute independently to generate feature vectors. Our framework combines word-embeddings and constraints to generate features.

Sidharth et al. apply embedding approaches and define aggregation functions to summarize terms within attributes [41]. They explore a series of current deep learning solutions for entity matching tasks, including *SIF, RNN, Attention* and *Hybrid*

Symbol	Description
R	Relation
$A_1, A_2...$	Relational attributes
X, Y	Sets of relational attributes
t	A tuple of R
$\|\mathcal{A}\|$	Total number of attributes
$\langle t, t' \rangle$	A pair of tuples
s	Term
w	Term weight
$f_1, f_2...$	Constraint features
$\overline{x}$	Feature vector
φ	Functional dependency (FD)
ω	Metric Functional dependency (MFD)
Δ	Tolerance parameter for MFD
Σ	A set of constraints

Table 3.6: Summary of notation and symbols.

approaches [41]. They evaluate the performance of these solutions on various entity matching (EM) problems, such as structured EM, textual EM and dirty EM [41]. However, a limitation of their approach is the extensive training time due to the complexity of the neural networks.

3.3 Preliminaries

We present preliminary definitions and notations used in this chapter.

3.3.1 Metric Functional Dependencies

Definition 3.3.1. A metric functional dependency (MFD) $\omega : X \xrightarrow{\Delta} Y$ holds if two tuples $t[X] = t'[X]$ then $d(t[Y], t'[Y]) \leq \Delta$, for a metric distance function d over numeric values in Y.

For a relation R, MFD defines a relationship between X and Y, where X and Y are sets of attributes in R [32]. MFD allowed difference between two tuples in Y should be within the given tolerance parameter Δ. The metric parameter Δ is defined as part of ω and normally set by users. Compared to FDs, MFDs relax the equality condition on the consequent Y attributes by allowing small differences. We can define the "closeness" of these values by a metric distance function d. Specifically, the metric of d satisfies four properties: (a) symmetry, which means $d(x, y) = d(y, x)$; (b) non-negativity, $d(x, y) \geq 0$; (c) identity of indiscernible, $d(x, y) = 0 \Leftrightarrow x = y$; (d) triangle inequality, $d(x, z) \leq d(x, y) + d(y, z)$ [32].

Another type of constraint, matching dependencies(MDs), extend FDs by relaxing the strictly matching to similarity measures [42]. In MDs, if two tuples are similar on attributes X ($t[X] \approx t'[X]$), then they should have equal value on attributes Y ($t[Y] = t'[Y]$) . The similarity measures in MDs need to calculate a similarity score in the range of $[0, 1]$, and compare this score to a given threshold. MDs state that only if the similarity score is greater than or equal to the threshold, then two tuples are considered as matching in attributes X ($t[X] \approx t'[X]$). However, the challenge here is to find a proper similarity threshold. This threshold parameter varies for different datasets, which makes it even harder to specify. Our model does not rely on any specific similarity threshold. Our model does not compare the similarity scores to the given threshold. Instead, our model converts the similarity scores to a feature vector

and uses this feature vector to train a classifier to identify duplicates.

3.3.2 Word-Embeddings

The *Word-embedding* approach can map words or phrases to a low-dimensional, real-valued vector for similarity calculation [43]. Each dimension of the embedding represents a feature of the word, which can capture useful syntactic and semantic properties. Unlike TF-IDF model, which tries to vectorize the word based on its frequency in the document and corpus, the word-embedding method tries to quantify and categorize semantic similarities between terms based on their context in the dataset.

Intuitively, if two words have similar "context", they should be similar as well. A widely used word-embedding model is skip-gram, which leverages the neighbouring words in the context of the target word to capture its semantics. To obtain the neighbouring words, skip-gram first partitions the given string into content pairs through a sliding context window and trains a neural network with these context pairs [44]. To obtain the accurate word vectors, the input context pairs should contain enough adjacent words for the neural network training process. In duplicate detection over relational attributes, attribute values do not always contain enough terms for a skip-gram partition, and some attribute even contains only a single term. To address this challenge, past work breaks the attribute boundaries and consider each tuple as a long string to apply word-embedding approaches [39, 41]. However, breaking attribute boundaries will lead to attribute and schema information loss in relational data, as the attribute associations are lost. We use the attribute relationships provided by integrity constraints to address this short text issue. Instead of assuming attributes are independent, integrity constraints allow us to leverage relationships among the

attributes.

3.4 Framework Overview

In this section, we formally define our problem, and then give an overview of our framework.

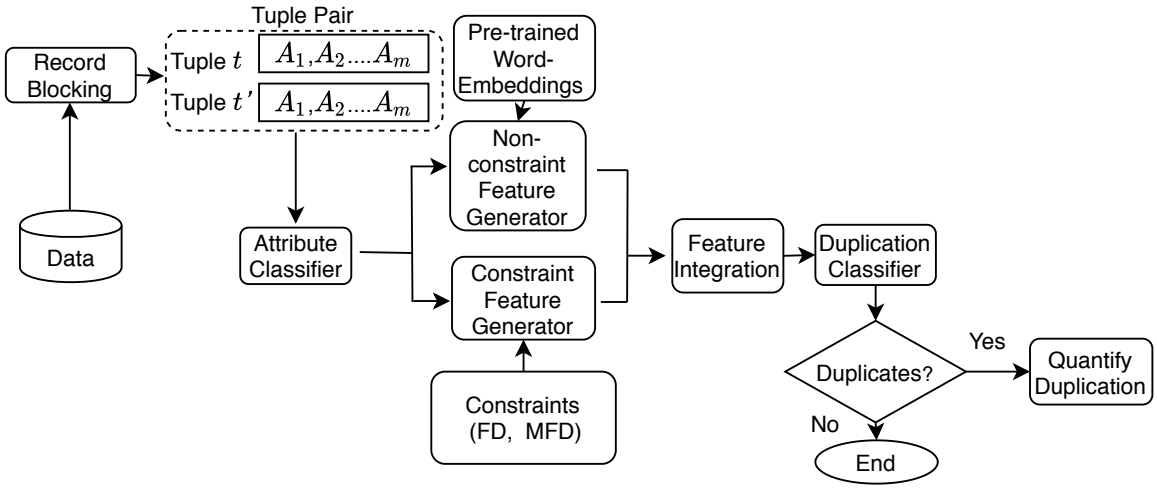


Figure 3.1: Deduplication Framework Overview.

3.4.1 Problem Statement

Our task is to identify all tuples that refer to the same entity in a relation R. Specifically, given two tuples $t, t' \in R$, they can form a pair of tuples $\langle t, t' \rangle$, and we need to determine whether $\langle t, t' \rangle$ is "matched" or "unmatched". We model this task as a binary classification problem, where we use labelled $\langle t, t' \rangle$ to train a classifier for deduplication. Since we cannot directly use a pair of tuples $\langle t, t' \rangle$ as input to the classifier, the challenge is how to construct a feature vector $\overline{x}$ to represent $\langle t, t' \rangle$. The problem can be formalized in the Equation 3.1.

$$\text{classify}(\langle t, t' \rangle) = \text{classify}(\overline{x}) = \begin{cases} \text{matched} \\ \text{unmatched} \end{cases} \tag{3.1}$$

3.4.2 Solution Overview

Figure 3.1 shows an overview of our deduplication framework. Given a dataset and a set of pre-defined constraints Σ, we first input the dataset into the record blocking module where tuples are partitioned into blocks to reduce runtime and overhead. We take a pair of tuples $\langle t, t' \rangle$ from a block, and categorize their attributes into two categories, *constraint attributes* or *non-constraint attributes* according to the given constraints Σ. Constraint attributes are the attributes that participate in any constraint, such as FDs and MFDs. Non-constraint attributes do not participate in any constraint. For non-constraint attributes, our framework converts attribute values to word-embedding vectors and automatically adjusts the weight of each word based on their frequency information to generate non-constraint features that we will describe in Section 3.5. For constraint attributes, our framework extracts the semantic information of the constraints to generate constraint features, which we will describe in Section 3.6. Finally, constraint features and non-constraint features will be integrated into a unified feature vector $\overline{x}$ to represent $\langle t, t' \rangle$, and it will be used as an input to the classifier.

Record Blocking

To identify duplicates of a given relation R with N tuples, existing approaches, which are based on the string similarity [4, 31, 33, 34, 35], have to calculate the similarity scores of tuples pairwise. The number of comparisons will be $O(\frac{N(N-1)}{2})$. This is

Algorithm 1: Record Blocking

Input : a set of tuples D

Output: A set of clusters C

$(s_1, s_2, ...s_k) \leftarrow SelectRandomSeeds(D)$
$C \leftarrow \{\}$
for $s_i \in (s_1, s_2, ...s_k)$ **do**
    $c_i \leftarrow \{s_i\}$
    $v_i \leftarrow CalculateTF - IDF(s_i)$
    for $t_j \in D$ **do**
        $v_j \leftarrow CalculateTF - IDF(t_j)$
        if $i \leftarrow argmin||v_i - v_j||^2$ **then** $c_i \leftarrow c_i \cup t_j$;
    end
    $C \leftarrow c_i \cup C$
end
return C

because the string similarity calculation is a binary operation, which only takes two strings as input and outputs one similarity score each time. Therefore, all of tuples in R have to compare with each other, which leads to $O(\frac{N(N-1)}{2})$ operations. To avoid a prohibitively expensive comparison for large datasets, we first partition tuples into disjoint blocks and compare tuples pairwise within each block only to reduce the overall number of comparisons. If we can partition N tuples into K blocks, then each block only contains $\frac{N}{K}$ tuples on average. The number of comparisons in each block will reduces to $O(\frac{N(N-K)}{2K^2})$. We can use a hash function to project similar tuples into blocks.

For example, Soundex [45] is a hash function which assigns tuples to phonetically similar groups based on the consonants. Another example is the weak hash function, but it is only suitable for a small dataset containing large files [46]. In our case, the tuples are small files which could lead to hash collisions in a large dataset. In our framework, we take the *k-means* [47] unsupervised learning algorithm to cluster the tuples into blocks. The *k-means* does not require labelled data nor any assumptions about the distribution of data. To cluster the tuples, we consider each tuple as a short document, and calculate the tf-idf score for each term that appears in the tuple. Each tuple $t \in R$ can be converted to a tf-idf vector.

The k-means algorithm partitions these vectors into k blocks to minimize these vectors' intra-cluster variance. As line 1-2 of Algorithm 1 shows, we first randomly select k tuples from D as the initial seeds $(s_1, s_2, ...s_k)$, and initialize C as the set of clusters. In line 3-4, We initialize k clusters for each seed $s_i \in (s_1, s_2, ...s_k)$, and compute their TF-IDF vectors. In line 5-8, we compute the TF-IDF for each tuple $t_j \in D$, and try to find the minimal intra-cluster variance between the vector of t_j and the vector of seed s_i to determine which cluster that t_j should be in. Once all tuples in D are partitioned into clusters, our algorithm will return the set of clusters. The intention is to reduce the complexity by reducing the number of tuple comparisons by grouping similar tuples together (per block). Before record blocking, there are N tuples in D, and the number of comparisons is $O(\frac{N(N-1)}{2})$. Algorithm 1 partitions N tuples into K blocks, then each block only contains $\frac{N}{K}$ tuples on average. The number of comparisons in each block will reduces to $O(\frac{N(N-K)}{2K^2})$.

Algorithm 2: Attribute Classify

Input : A pair of tuples $\langle t, t' \rangle$, a constraint φ

Output: A set of non-constraint attributes **P** and a set of constraint attributes **U**

```
1 A ← ⟨t, t'⟩
2 P ← ∅
3 U ← ∅
4 for A_i ∈ A do
5 |   if A_i ∈ φ then U ← U∪ A_i ;
6 |   else P ← P∪ A_i ;
7 end
8 return U, P;
```

Attribute Classifier

Existing approaches break the boundaries of attributes and treat tuples as long string to calculate the string similarity scores for deduplication [34, 35]. However, they do not consider the relationships among attributes, which may contain useful information that can be used for deduplication. We use the pre-defined constraints to capture the semantic relationships among attributes, and propose a set of statistical features to model these relationships. Since not all attributes participate in the pre-defined constraints, we use word-embeddings and similarity functions to generate features for the attributes that do not participate in any constraints. We partition the attributes into two categories and propose different approaches to generate their features. Given

a constraints φ, we divide the attributes $A_1, A_2, ...$ of the relation R into two categories: non-constraint attributes set **P** and constraint attributes set **U**. From line 1 to line 3 of the Algorithm , we obtain all attributes **A** from $\langle t, t' \rangle$, and initialize the non-constraint attributes set **P** and the constraint attributes set **U**. In line 4-7, we partition the attributes based on the given constraint φ. For an attribute $A \in \mathbf{A}$, if A participates in the given constraint φ, A will be partitioned into the set of constraint attributes **U**; otherwise, A will be partitioned into the set of non-constraint attributes **U**. The constraint features use constraints to measure the semantic relationship among attributes and the non-constraint features leverage the similarity functions to compute the similarity scores on the matching attributes. They model two tuples from different perspectives to extract the representation features of $\langle t, t' \rangle$. Hence, we combine them into a unique feature vector to train the deduplication classifier.

Feature Generation

In this component, we use different feature generators to extract the features from the set of non-constraint attributes and the set of constraint attributes. The non-constraint attributes set **P** and constraint attributes set **U** will be used as input for this component. For the set of non-constraint attributes, we use three different approaches (edit distance, q-grams and word-embeddings) to generate features. For a set of constraint attributes, we use the constraint among attributes to generate constraint features. We will discuss non-constraint and constraint feature generation in Section 3.5 and 3.6, respectively.

Feature Integration

Algorithm 3: Feature Integration

Input : A non-constraint feature vector $\overline{x'}$ and a constraint feature vector $\overline{x}$

Output: A integrated feature vector $\overline{z}$

1 $\overline{z} \leftarrow \emptyset$

2 $\overline{z} \leftarrow \overline{x'} \oplus \overline{x}$

3 **return** $\overline{z}$;

The feature integration module will integrate the non-constraint features and the constraint features into a unified feature vector to represent the pair of tuples $\langle t, t' \rangle$. Since the order of features has no impact on the classification result, we can integrate these two feature vectors by concatenating them. Assume vector $\overline{x} = [x_1, x_2, ...x_h]$ represents the non-constraint feature vector of $\langle t, t' \rangle$, and $\overline{x'} = [x'_1, x'_2, ...x'_l]$ represents the constraint feature vector of $\langle t, t' \rangle$. Symbol $\oplus$ represents the feature integration operator, thus $\overline{x} \oplus \overline{x'} = [x_1, x_2, ...x_h, x'_1, x'_2, ...x'_l]$. This integrated feature vector will be used as the input of the classifier.

3.5 Non-constraint Attributes Feature Generation

Identifying duplicates includes two tasks: (i) extract the features of tuples; (ii) classify the tuples to "match" or "unmatched" category. In this section, we discuss how to extract features from non-constraint attributes. We first present two distance functions for similarity computation, and then present a frequency-based approach for term differentiation.

3.5.1 Edit Distance

Edit distance computes the minimum cost of transforming one string to another string based on counting the number of necessary edits [31]. The edit distance between two terms s and s' is the minimum number of edit operations (of single characters) needed to transform s into s'. To obtain the similarity score, we have to normalize the edit distance into the range $[0.0, 1.0]$. These similarity scores will not be used to compare against a given threshold to determine duplicates. Instead, these similarity scores will be used as feature vectors and integrated with the constraint feature vectors to train the classifier. Our model does not rely on any specific similarity threshold to compare against, which makes our model more flexible than existing string similarity approaches [34, 35].

We define normalized similarity score between terms (s, s') as:

$$sim(s, s') = 1 - \frac{editDist(s, s')}{max(|s|, |s'|)} \tag{3.2}$$

$editDist(s, s')$ denotes the edit distance of (s, s'), and $max(|s|, |s'|)$ represents the maximal length between s and s'.

Example 3.5.1. *Given two attribute values representing company names, AllianzGI Convertible Income Ltd and AllianzGI Convert Income Corp, we can split them into terms to obtain [(AllianzGI,AllianzGI) (Convertible,Convert) (Income,Income) (Ltd,Corp)]. The edit distances of these term pairs are* $[0, 4, 0, 4]$*, which can be normalized as* $[1-0, 1-4/11, 1-0, 1-4/4] = [1, 7/11, 1, 0]$*. These normalized values will be used as a feature vector* $\overline{x} = [1, 7/11, 1, 0]$ *to integrated with the constraint feature*

vector $\overline{x'} = [2/3, 1/2, 1, 0]$ *through the integration operator* $\oplus$*. In this way, we can obtain the integrated feature vector* $\overline{x} \oplus \overline{x'} = [1, 7/11, 1, 0, 2/3, 1/2, 1, 0]$*. This integrated feature vector along with the given label data will be used to train the classifier.*

3.5.2 q-grams Similarity

In q-grams similarity [48], each attribute value will be converted to a set of q-grams, and the comparison between two tuples is based on their q-grams sets. The intuition is that if two strings are similar, they should share a large number of q-grams. To generate a q-grams set, we consider a sliding window of size q over term s, and use the sequence of characters within the window as a token. In this thesis, we use $q = 2$ to generate bigram tokens.

Example 3.5.2. *Consider the three terms* $s_1 = patent$*,* $s_2 = part$*,* $s_3 = apaten$*, with the following bigram sets:*

$$bigrams\ of\ s_1 = \{pa, at, te, en, nt\}$$

$$bigrams\ of\ s_2 = \{pa, ar, rt\}$$

$$bigrams\ of\ s_3 = \{ap, pa, at, te, en\}$$

To calculate the bi-gram, we use a bitmap, such as Table 3.7. Bitmaps allow us to efficiently compare tuples, and calculate the similarity score. For two bitmaps, we use the bitwise AND and OR operation to compare bits, to determine whether a bigram appears in a tuple. We then use Jaccard similarity [48] to calculate the similarity score between the bitmap representations of s and s', namely, b_s and $b_{s'}$, respectively.

	pa	at	te	en	nt	ar	rt	ap
patent	1	1	1	1	1	0	0	0
part	1	0	0	0	0	1	1	0
apaten	1	1	1	1	0	0	0	1

Table 3.7: bigrams bitmap

$$sim(s, s') = \frac{b_s \cap b_{s'}}{b_s \cup b_{s'}} \tag{3.3}$$

where b_s is the bitmap representation of s, and $\cap$ denotes bitwise AND operation, and $\cup$ denotes the bitwise OR operation. Similarly, the similarity scores will be used as feature vectors to train the classifier as the edit distance we discussed above.

3.5.3 Term Differentiation

Existing techniques [34, 35] have primarily assumed that all the terms have equal weight but ignore the difference between infrequent terms and frequent terms during the comparison. For example, given two values *IBM Corp* and *ABB Corporation*, terms *IBM* and *ABB* occur less frequently but more distinguishable than *Corporation* and *Corp*. Therefore, we should consider their contributions to the similarity score differently. In our framework, we distinguish these less frequent terms during the aggregation process by placing more emphasis on the infrequent terms (such as *IBM* and *ABB*) since they serve as a more accurate indicator of similarity than commonly occurring terms in the data. Intuitively, if tuples have a high similarity score on the common and insignificant term such as *Corporation* they will offset the difference between their core term *ABB* and *IBM*.

We use w to denote the weight of term pair (s, s'), which can be calculated through their term frequency $w = -\log \frac{freq(s)+freq(s')}{2}$. The intuition is that the weights of terms with high frequencies should be lower than the terms with low frequencies. In the next section, we will discuss how to use these weights to refine our similarity function.

3.5.4 Weighted Frequency Similarity

We combine the similarity scores with term differentiation weights to refine the non-constraint attribute features. We assume the terms in each attribute are aligned, and we can couple the similarity score of each term $sim(s_i, s'_i)$ with their corresponding weight w_i to obtain the similarity score of $\langle t, t' \rangle$ on the attribute A (denoted as $sim(t[A], t'[A])$). Similarly, we can apply this approach to other attributes of $\langle t, t' \rangle$ and obtain the similarity scores of other attributes. All of these attribute similarity scores will be considered as features and integrated into a unique feature vector through the $\oplus$ operator such as $sim(t[A_1], t'[A_1]) \oplus sim(t[A_2], t'[A_2])$....

$$\begin{aligned} sim(t[A], t'[A]) &= \sum_{i \in n} w_i \cdot sim(s_i, s'_i) \\ &= \sum_{i \in n} (-\log \frac{freq(s_i) + freq(s'_i)}{2}) \cdot sim(s_i, s'_i) \end{aligned}$$

We can apply this w_i to the adapted Edit Distance similarity function and obtain the *Weighted Frequency Edit Distance (WFED)* similarity score:

$$WFEDsim(t[A], t'[A]) = \sum_{i \in n} w_i \cdot sim(s_i, s'_i)$$
$$= \sum_{i \in n} w_i \cdot (1 - \frac{editDist((s_i, s'_i)}{max(|s_i|, |s'_i|)})$$

where $w_i = -\log \frac{freq(s_i)+freq(s'_i)}{2}$.

Similarly, we define the *Weighted Frequency Bigrams (WFB)* similarity score by coupling w_i to the bigrams similarity function:

$$WFBsim(t[A], t'[A]) = \sum_{i \in n} w_i \cdot sim(s_i, s'_i)$$
$$= \sum_{i \in n} w_i \cdot \frac{b_{s_i} \cap b_{s'_i}}{b_{s_i} \cup b_{s'_i}}$$

where $w_i = -\log \frac{freq(s_i)+freq(s'_i)}{2}$.

3.6 Constraint Attributes Feature Generation

Given a set of pre-defined FDs and MFDs, we leverage them to compare the values among attributes participating in the constraint. We begin by extracting a set of features $\{f_1, f_2, f_3, f_4\}$ that capture statistics about the attributes and consistency of the constraints over the data.

3.6.1 Constraint Properties

Not all FDs and MFDs work well in the deduplication task. In this chapter, we adopt a different notion of constraint satisfaction where the FDs (denoted as φ) and MFDs (denoted as ω) are evaluated on a pair of tuples $\langle t, t' \rangle$, i.e., $\langle t, t' \rangle \models \varphi$, $\langle t, t' \rangle \models \omega$. Our framework uses constraints as a proxy of attribute relationships that should hold over the tuples and we check a pair of tuples $\langle t, t' \rangle$ at a time. FDs are hard constraints which requires t and t' to match on X and Y. MFDs generalize FDs with a tolerance parameter Δ, which allows them to be more flexible than FDs, but they are still hard constraints such that the difference of t and t' on the Y must lie within Δ. To ensure that we can check as many attributes over $\langle t, t' \rangle$ as possible, we seek FDs and MFDs with the following properties: (a) where the antecedent (LHS) attributes of φ and ω, all these attributes together are a maximal number (with the maximum being the number of attributes in R). (b) constraints containing pseudo-identifiers that are close to keys (identifiers) because these pseudo-identifiers help to uniquely identify entities. If the pseudo-identifiers of two tuples are exact matching, these two tuples are duplicates with high likelihood. (c) constraints containing few overlapping attributes because the classifier may overestimate the overlapping attributes.

3.6.2 The Number of Satisfied Constraints

Let Σ be a set of constraints that are either FDs (φ) or MFDs (ω). For ease of presentation, we use φ here, but all features apply to ω as well. Let $S = \{\varphi \in \Sigma | \langle t, t' \rangle \models \varphi\}$ be the set of all constraints that are either FDs or MFDs, whereby t, t' satisfy φ. We define feature f_1 as the number of satisfied constraints i.e., $|S|$, since it provides an indicator of the overall similarity between t and t'. Therefore, we

represent f_1 as below:

$$f_1 = \frac{|S|}{|\Sigma|} \tag{3.4}$$

3.6.3 Matching Constraint Attributes

For $\langle t, t' \rangle$, the number of attributes that matches a constraint provides a measure of equality (in the case of φ), or similarity (in the case of the consequent attributes in ω). The number of attributes that are considered equivalent is a useful indicator of whether t and t' are duplicates. Hence, we define a feature, f_2, which counts the number of attributes participating in the constraints by $\langle t, t' \rangle$, and normalize it with the total number of attributes in R. For $\varphi : X \rightarrow Y$ or $\omega : X \xrightarrow{\Delta} Y$, where $\langle t, t' \rangle \models \varphi$ or $\langle t, t' \rangle \models \omega$, let $|XY|$ represent the number of attributes in φ, ω and $|\mathcal{A}|$ represent the total number of attributes, then the f_2 feature can be denoted as:

$$f_2 = \frac{|XY|}{|\mathcal{A}|} \tag{3.5}$$

3.6.4 Proportion of Exact Matching Attributes

The proportion of exact matching attributes can be used as a good indicator of whether t and t' are in duplicates. Since both FDs and MFDs require exact matching on X, the more attributes on X indicate high proportion of exact matching attributes between t and t'. For example, if two tuples have exact values on X attribute it indicates that two tuples are probably duplicates. We define a feature f_3 to capture the proportion of exact matching attributes in X relative to $|XY|$ in the constraints. Intuitively, we are trying to quantify the proportion of exact matching attributes in

X, but we do not currently enforce the dependency semantics to check for equality (or similarity in the case of MFDs) in Y. We compute the ratio of $|X|$ over $|XY|$ for all satisfied $\varphi, \omega \in \Sigma$. Specifically, for $\varphi : X \rightarrow Y$, (similarly for ω), if $\langle t, t' \rangle$ satisfies $\{\varphi, \omega | \langle t, t' \rangle \models \varphi \vee \langle t, t' \rangle \models \omega\}$, we defined f_3 as:

$$f_3 = \frac{|X|}{|XY|} \tag{3.6}$$

Since both FDs and MFDs require exactly matching on the X, $|X|$ indicates the number of exactly matching attribute values between the two tuples, which is an indicator of their likelihood to be duplicated.

3.6.5 MFD Tolerance Parameter

An MFD $\omega : X \xrightarrow{\Delta} Y$ allows minor difference in Y by specifying a tolerance parameter Δ, and Δ serves as an upper bound of the allowed difference in Y. Given $\langle t, t' \rangle$, if $\langle t, t' \rangle \models \omega$, then the actual difference of $\langle t, t' \rangle$ on Y (denoted as $\Theta = |t[Y] - t'[Y]|$) should be less than or equal to the given upper bound Δ. The value of Θ indicates the closeness of $\langle t, t' \rangle$ on the Y. Hence, we define f_4 to compute the actual difference Θ and normalize it with Δ. Since there could be multiple pre-defined MFDs with different Δ_i and Θ_i, we sum these values in f_4:

$$f_4 = \sum_i \frac{\Theta_i}{\Delta_i} \tag{3.7}$$

3.7 Duplication Metric

In this section, we define duplication metrics to quantify the level of duplication for a single attribute value, and the duplication level across an attribute domain. These metrics can help users to better understand the distribution of similarity values and guide the data cleaning process.

3.7.1 Duplication of an Attribute Value

During tuple comparisons, we store the similarity scores between terms. For a term s, we can model the related terms (satisfying a given similarity threshold) as a graph, where the set of vertices $\mathcal{V}$ represent terms and edges $\mathcal{E}$ represent the similarity score between two terms.

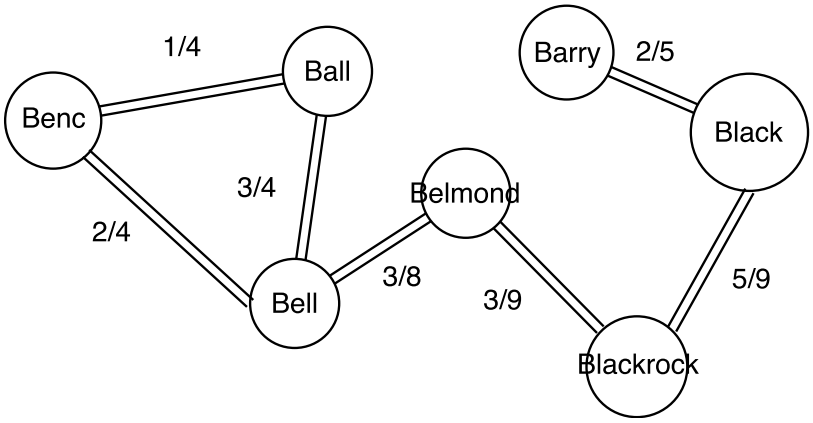


Figure 3.2: Similarity graph for an attribute value.

Given a vertex v, let $\mathcal{E}_v$ denote its edge set and $|\mathcal{E}_v|$ denote the size of $\mathcal{E}$. Let edge $edge_{vw} \in \mathcal{E}_v$ represent the similarity score between terms represented by nodes v and w.

$$dup(v) = \frac{\sum_w e_{vw}}{|E_v|} \tag{3.8}$$

The duplication score, $dup(v)$, represents the average similarity of v to its neighbouring values. Higher dup scores indicate a larger number of similar values (duplicates) to v, on average. In contrast, lower dup scores indicate less similar (distinct) neighbouring values to v. Figure 3.2 shows the similarity graph for company name *'Bell'*.

3.7.2 Duplication across an Attribute Domain

Figure 3.3 shows the similarity graph for a sample of domain values in an attribute *'company name'*. At an attribute (column) level, we consider all values in the domain, and cluster values that likely refer to the same entity. From the similarity graph *graph*, we identify all articulation points in *graph*, which separate nodes into clusters based on the bi-connected components. An articulation point in a graph is the vertex that its removal makes the graph disconnected. Intuitively, we can think of articulation points as "connector" nodes that link two or more clusters, each representing a distinct entity. For each articulation point a, we compute a duplication score $dup(c)$ for each cluster c that a is connected to, as the average of the edge weights between a and other nodes in the cluster c.

$$dup(c) = \frac{\sum_{v \in c} e_{av}}{deg(a)} \tag{3.9}$$

where $deg(a)$ is the degree of node a. The score $dup(c)$ measures the closeness between values (represented by nodes) in the cluster c, and the value (represented

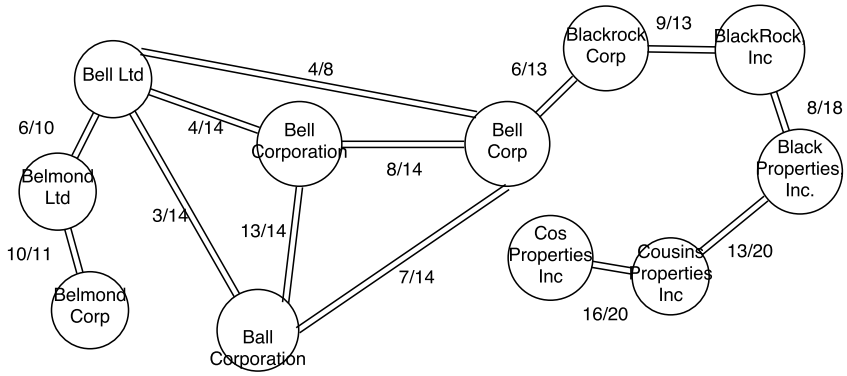


Figure 3.3: Similarity graph for an attribute domain.

by node) a. To compute the duplication score for an attribute A, we compute the average $dup(c)$ score across all clusters $c \in C$.

$$dup(A) = \frac{\sum_c dup(C)}{|C|} \tag{3.10}$$

Example 3.7.1. *In Figure 3.3, consider articulation points 'Bell Ltd', and 'Bell Corp'. We have four clusters, as shown by the coloured edges. Consider the cluster marked by edges in green. We compute* $dup(u_1) = (\frac{4}{8} + \frac{4}{14} + \frac{3}{14})/3 = 0.33$. *Similarly, we get* $dup(u_2) = 0.6$, $dup(u_3) = \frac{6}{13}$, $dup(u_4) = avg(\frac{8}{14} + \frac{7}{14})$, *and obtain* $dup(A) = avg(0.33 + 0.6 + 0.46 + 0.53) = 0.48$.

3.8 Limitations

In this section, we discuss the limitations of our framework, and provide potential solutions to be explored as future work.

Handling False Positives and False Negatives.

Our model cannot achieve 100% accuracy, which means some identified duplicates are non-duplicate (false positives). False positives may occur when two tuples satisfy

the pre-defined constraints but they refer to different entities. False negatives occur when two tuples are duplicates, but our model did not identify them as duplicates. This may occur when two tuples did not satisfy any pre-defined constraints and our model cannot extract any constraint features from them.

False positive and false negatives can occur due to limitations in the learning model which require extracting abstract features from the data and constraints. Obviously, this feature extraction abstraction may have some information loss, which could lead to false positive and false negative in the results. The f_1 feature measures the ratio between the number of satisfied constraints $|S|$ and the number of constraints $|\Sigma|$. However, satisfying the pre-defined constraints only indicates that two tuples are matching on the attributes participating in the constraints, and it does not mean they are matching on all attributes of R. Therefore, even two tuples satisfy all pre-defined constraints, it does not indicate they are duplicates since they may not match on the non-constraint attributes. The f_2 feature measures the number of attributes participating in the constraints. It is possible that two tuples satisfy multiple constraints, and these constraints contain overlapping attributes. In this case, the f_2 feature will provide a stronger signal due to overlapping attributes, thereby providing an overestimation that the two tuples are duplicate. One way to correct this is to avoid double counting the overlapping attributes. An alternative is to provide a priority mechanism via weights to favour particular attributes participating in the constraint.

The f_3 and f_4 features measure the proportion of exact matching attributes and the ratio between actual difference and the tolerance parameter, respectively. The f_3 feature relies on the number of attributes participating in the constraints as well. If

the exact matching attributes (X) was only a small portion of all attributes, it only indicates two tuples matching on a few attributes, which cannot determine whether they are duplicates. The f_4 feature relies on the value of tolerance parameter Δ. Our framework prefers MFDs with small Δ values, otherwise our framework may produce inaccurate results. For example, if users specify an improper large Δ, the f_4 feature may misidentify two tuples with large difference as duplicate.

One potential solution to address the limitations in our constraint features, $f_1 - f_4$ is to include the power of the crowd [49]. This provides a more accurate but expensive way to bring human experts into the deduplication process. In our case, we can define a two-stage machine-human hybrid workflow to identify duplicates. The first stage is a machine-based approach, which still leverages the machine learning algorithm to identify duplicates. Instead of making the final classification decision, the machine learning algorithm will provide confidence scores for tuples to indicate the likelihood of duplication. In the second stage, only those tuples with low confidence scores will be sent to human experts for examination and verification. In this way, we can dramatically improve accuracy.

Constraint Quality

All of these constraint features provide signals to detect duplicates, if the quality of the given constraints is low, it could lead to false positive results. Since the accuracy of our model relies on the quality of pre-defined constraints, one way to address this issue is to mine the constraints from the given dataset instead of relying on the pre-defined constraints. There are many constraints mining techniques to discover good FDs from the given dataset. For example, FDMine is a constraint mining technique

which uses Armstrong's Axioms to mine constraints from data [50]. Another solution is to assign a penalty to the constraint features that have negative impact to the results. If some features cause too many false positives, we can decrease the weight of that feature to reduce the impact of "bad" FD on the deduplication results.

Adversarial Samples

Adversarial samples are data points with malicious, incorrect labels in the training dataset, and they aim to deceive and mislead the learning model to make a false prediction. As our framework uses the supervised machine learning model to detect duplicates, our model is also vulnerable to adversarial attacks. Attackers can inject malicious samples into our training dataset by providing incorrect labels. For example, they can deliberately mark true duplicates as "unmatched" but mark irrelevant tuples as "matched" to mislead our classifier. In the machine learning domain, one common approach to address adversarial samples is to perform adversarial training [51].

In adversarial training, many adversarial samples are intentionally generated and injected into the training data, and used to train a more robust model. In our framework, we may consider injecting tuples with incorrect labels, and impose penalties via parameter re-weighting to rectify these samples in the next learning iteration. The goal is to train a robust model that is not sensitive to adversarial samples.

Runtime Improvement

Our framework needs to compute the similarity between tuples pairwise which leads to $O(N^2)$ runtime in worst case. A potential solution for improving the runtime is to use blocking or clustering techniques to group similar tuples [52]. Similar tuples are clustered into blocks and pair-wise comparisons are executed only within each

block. For example, we can apply weak hash or K-means algorithm to group the similar tuples, which can reduce the number of comparisons in each block. Another potential solution is to run pairwise comparison parallelly. We can deploy our framework in a distributed environment such that the comparison in each block can run simultaneously.

3.9 Evaluation

We turn to the evaluation of our techniques, and conduct an extensive set of experiments to compare the accuracy and performance of our techniques with existing techniques. Our evaluation focuses on these objectives:

1. The effectiveness of our term frequency weighting, and the inclusion of data integrity constraints as part of the deduplication task.

2. The comparative accuracy of our model by coupling our term frequency weighting with three different similarity functions (edit distance, bi-gram similarity, and word-embeddings).

3. Evaluating the efficiency and effectiveness of our model as we scale a set of parameters.

4. The comparative accuracy of our model over existing ER solutions, including traditional solutions WHIRL [33], SERF (Stanford Entity Resolution Project) [35] , and state-of-art learning-based techniques such as Magellan [40], and Hybrid [41].

3.9.1 Experimental Setup

We implement our framework in Python 3.6, and test on a server with Intel Xeon 2.2 GHz processor and 64GB of RAM on Linux system. We split all of the datasets into three parts 70%:10%:20% for training, validation and testing, respectively. In our experiments, we apply four different machine learning models (SVM, Logistic Regression, Random Forest, Naive Bayes) to the datasets and report the average precision and recall of these techniques. For constraint attribute values, we use the statistical features proposed in Section 3.6 to capture the relationship between attributes. For non-constraint attribute values, we use our Weighted Frequency Bigram (WFB), and Weighted Frequency Edit-Distance (WFED) approaches, as discussed in Section 3.5. We also use the pre-trained word-embeddings with our constraint features and call this method (WE). Note that these three approaches use the same procedure to calculate the features of constraint attributes, but use different methods to calculate the features of non-constraint attributes. For the word-embeddings approach (WE), we use the Google pre-trained word-embedding vectors [53, 54], which includes word vectors for a vocabulary consisting of three million words from large corpora, such as Wikipedia, GoogleNews, etc. We use the pre-trained word-embedding as a lookup table for word vectors: given a word, we can check the word-embedding corpus to obtain the vector of this word. To compute the similarity score of two words, we first check their vectors separately in the word-embedding lookup table, then compute the similarity score between these two vectors. Most common words can be found in this three million word-embedding corpus. For rare words that are not included in that corpus, we use edit distance to compute their similarity score directly. In our framework, since the word-embeddings are pre-trained and ready to use, we do not

consider the word-embeddings training time in our runtime results.

Datasets. We use six real datasets in our evaluation. These datasets are benchmark datasets, which are widely used for entity resolution.

- **Restaurants** [55]. This dataset contains California restaurant information such as names, addresses, and category, etc . We define two FDs on this dataset: [address, name] $\rightarrow$ [category] and [address, city] $\rightarrow$ [phone].
- **DBLP-ACM** and **DBLP-Scholar** [56]. Both of them describe bibliographical information of authors and their publications, such as title, authors, venue and year. We define one FD on these two datasets, [title, authors, year] $\rightarrow$ [venue].
- **Abt-Buy** and **Amazon-Google** [56]. These two datasets contain product information, including product name, description, manufacturer information and price. These two datasets have the same schema, we define one MFD [product name] $\xrightarrow{100}$ [price] on both of them.
- **Stocks [57].** This dataset is from the NASDAQ stock exchange, and contains the stock information such as company name, stock symbol, market capital, industry, sector, IPO date, etc. We define one FD on it, [company name, sector, industry] $\rightarrow$ [stock symbol].

The details of these datasets are given in Table 3.8. # Attr. means the number of attributes, Size means the number of tuples, and Matched means the number of duplicate tuples in each dataset.

Comparative Baselines. We compare our model with existing ER solutions, which includes traditional solutions as well as the state-of-art learning-based techniques.

Name	# Attr.	Size	Matched
Restaurants	5	864	112
DBLP-ACM	4	12,363	2,220
DBLP-Scholar	4	28,707	5,347
Abt-Buy	4	9,575	1,028
Amazon-Google	4	11,460	1,167
Stocks	12	14,744	1,382

Table 3.8: Experiment datasets

Method	Precision	Recall
ConstraintDedup [8]	0.82	0.78
SERF [35]	0.65	0.67
WHIRL [33]	0.62	0.57
Magellan [40]	0.73	0.70
Hybrid [41]	0.86	0.80

Table 3.9: Comparison Summary

WHIRL [33]. It uses TF-IDF weights with cosine similarity to calculate string similarity for identifying duplicates. It divides a string into a set of terms, and calculate TF-IDF for each term. Given this term representation, the tuple is modelled as a vector containing TF-IDF scores for each term. The cosine similarity is computed based on the TF-IDF vector to determine duplicates.

SERF [35].The Stanford Entity Resolution Framework (SERF) identifies duplicates via a matching phase followed by a merging phase. They use a matching function to calculate the similarity score of attribute values, and compare it with a given threshold. If the score is greater than the threshold, they merge these two tuples into a consolidated tuple representing a unique entity.

Magellan [40].Magellan is another learning-based framework that builds a pipeline to deal with duplicates. Magellan also treats deduplication as a binary classification and converts tuple attributes to features to train the classifier. It treats each attribute value as an independent string, and use the string similarity as the feature vector.

Hybrid [41].This technique uses word-embedding ad aggregation function to summarize terms within attributes. It applies deep learning techniques to the entity matching process. They categorize the entity matching (EM) problems into structured EM, textual EM and dirty EM, and define a design space for each category.

3.9.2 Comparative Accuracy: Term Differentiation and Similarity Functions

We evaluate the accuracy of WFB, WFED, and WE over six datasets. Figure 3.6 and Figure 3.7 show the precision and recall on six datasets. We observe that after applying weighted frequency, WFB achieves comparative precision and recall on

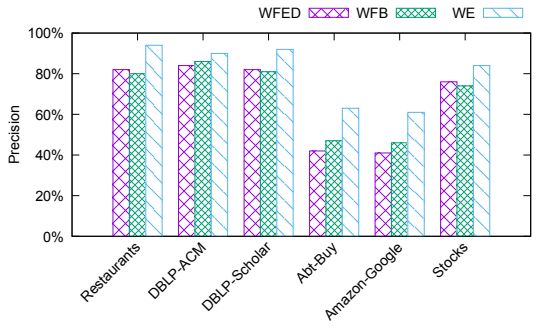


Figure 3.4: Comparative precision.

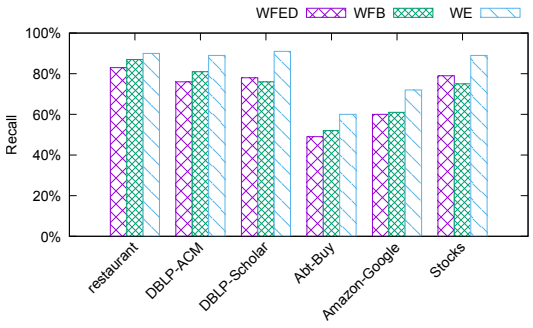


Figure 3.5: Comparative recall.

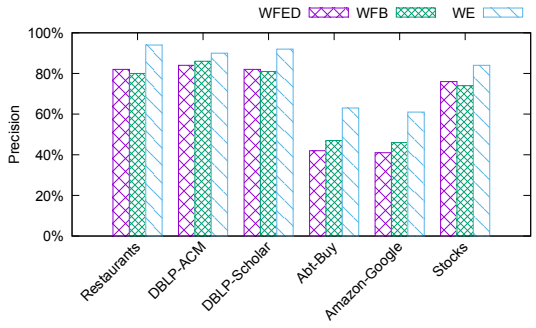


Figure 3.6: Comparative precision.

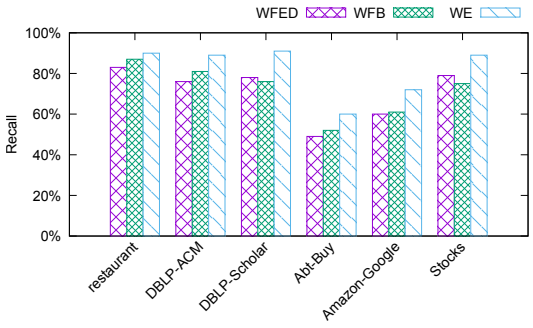


Figure 3.7: Comparative recall.

Restaurant, Stocks, DBLP-ACM and DBLP-Scholar datasets. The precision results achieve modest gains on Abt-Buy and Amazon-Google datasets because the records in Abt-Buy and Amazon-Google are commercial products records, which contain many synonyms such as *computer* and *laptop*, *mobile* and *telephone*, etc. These synonyms make it difficult for the distance-based approaches (e.g. WFB and WFED) to identify duplicates. However, WFB still can achieve comparatively larger precision gains than WFED due to finer-grained bigram matching for each term. The average gain of WFB is +5% in precision and +7% in recall. WE achieves improved results (+16% in precision and +13% in recall), especially on Abt-Buy and Amazon-Google due to its ability to recognize domain semantics, but requires pre-trained word-embedding vectors from an external source.

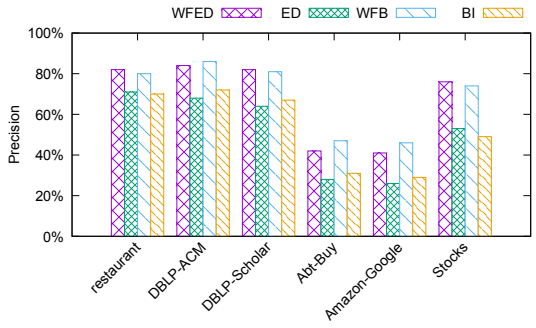


Figure 3.8: Term diff: precision.

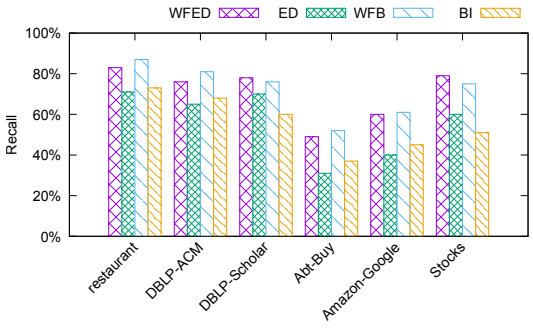


Figure 3.9: Term diff: recall.

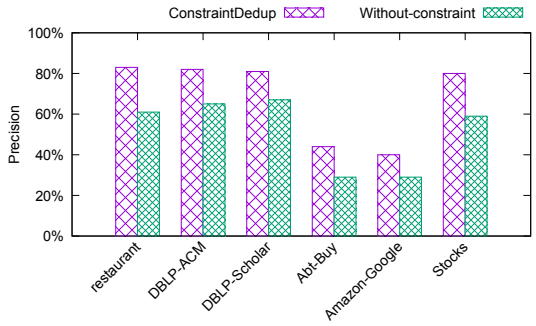


Figure 3.10: Constraints: precision.

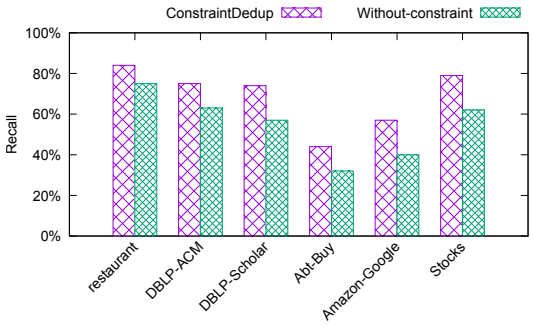


Figure 3.11: Constraint: recall.

3.9.3 Weighted Term Frequency Efficiency

To evaluate the efficiency of our weighted term frequency approach on ER, we compare our weighted frequency edit distance (WFED) with normal edit distance (ED), as well as weighted frequency bigrams (WFB) with normal bigrams (BI) approach. Figure 3.8 and 3.9 show the precision and recall respectively of applying these four approaches on six datasets. As we expect, both precision and recall improves under the weighted term frequency approaches (WFED and WFB). We observe that by using weighted term frequency in entity resolution, we achieve an average +13% and +16% increase in precision and recall, respectively.

3.9.4 Utility of Constraint Features

To evaluate the utility of our constraint attribute features, we compute the precision and recall of our approach ConstraintDedup (the word-embedding approach with constraint features) against the same approach but remove the constraint attribute features. Figure 3.10 and 3.11 show the precision and recall respectively over the six datasets. As expected, leveraging semantic relationships among attributes (if known) allows our technique to achieve an average +12% and +16% gain in precision and recall, respectively.

3.9.5 Varying the Size of Training Data.

A good learning-based approach should be robust to identify duplicates with as small training size as possible. To investigate the effectiveness of our framework, we vary the size of the training samples to evaluate its impact on accuracy. We choose two datasets: structured DBLP-Scholar dataset and semi-structured Amazon-Google dataset. To avoid over-fitting, we take random samples from the given datasets as the training data. Every time we vary the size of the training dataset, we re-train our classifier based on new samples. Figure 3.12 and Figure 3.13 show the precision and recall as we vary the training size on the DBLP-Scholar dataset. Figure 3.14 and 3.15 show the precision and recall on Amazon-Google. As we expect, both precision and recall improves as the number of training samples increases. The WE method gets better precision and recall than the other two approaches due to its stronger semantic representation. WFB and WFED can provide a competitive performance against WE on the structured DBLP-Scholar dataset. When the training samples increases to

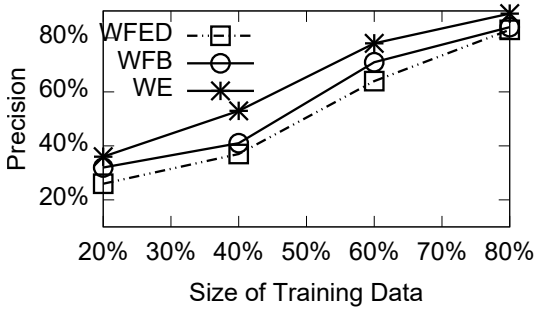


Figure 3.12: Precision (DBLP-Scholar).

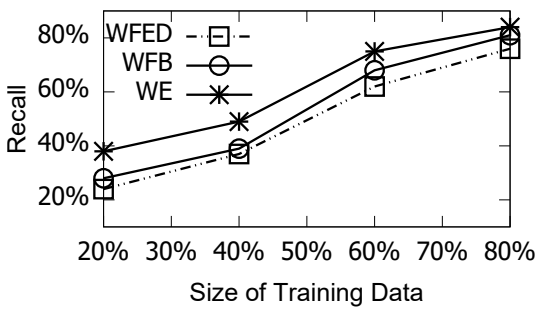


Figure 3.13: Recall (DBLP-Scholar).

80%, all three approaches obtain close to 81% precision and 78% recall on DBLP-Scholar dataset because the tuples in this dataset are more structured than the other dataset. The gain of WE is 20% more than the other two approaches on average on Amazon-Google dataset because of its strong semantic representation on synonyms for the Amazon-Google dataset. As the size of the training samples decreases, the accuracy of our model decreases as well. For example, with 40% training samples, our model only obtains 54% precision and 50% recall. Since our model is based on supervised learning techniques, and it relies on the training dataset to learn the features. If the size of the training dataset is small, our model cannot learn enough features to deal with unseen data, which leads to low accuracy. In the next step, we plan to study the learning techniques with few samples, such as transfer learning and reinforcement learning [58, 59] , and extend our framework to include these techniques.

3.9.6 Statistical Feature Analysis

To investigate the impact of our proposed statistical features on the accuracy, we analyze the weights learned by a logistic regression (LR) classifier. In the LR learning

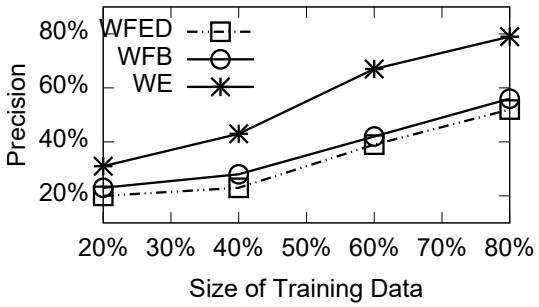


Figure 3.14: Precision (Amazon-Google)

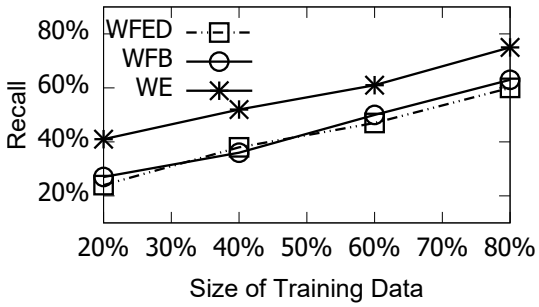
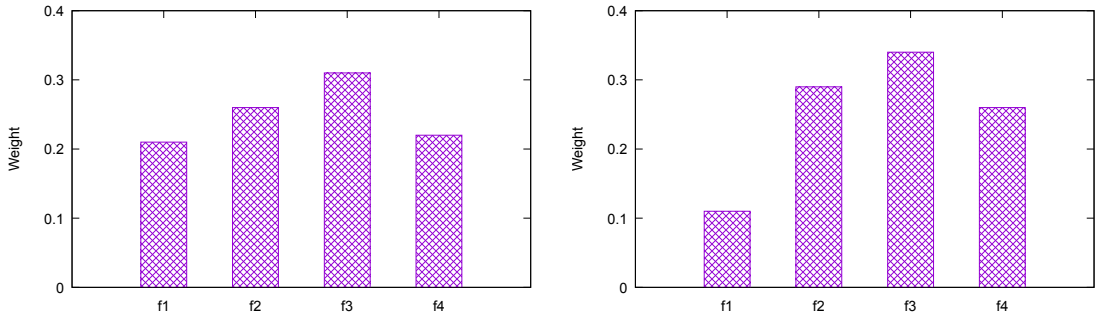


Figure 3.15: Recall (Amazon-Google)

process, each feature will be assigned an initial weight. During the training process, these weights are re-calculated to minimize the loss function, which is the difference between the predicted value and actual value through gradient descent optimization. When the optimization converges, the weight of each feature indicates its relative contribution towards predictive accuracy. Figure 3.16a and 3.16b show the relative weights of the features over the Abt-Buy and Amazon-Google datasets, respectively. The magnitude of the weights indicates the strength of the relationship between the statistical feature and its relevance to the prediction results. We can see that f_3 has the greatest weight, and hence, the greatest contribution towards accuracy, across all datasets. This indicates 0roportion of exact matching attributes has a strong impact on the duplications results. The large value of f_2 indicates that the number of matching constraint attributes is also a strong signal to predict the duplicate tuples. It is because both FDs and MFDs have strictly matching requirement on the left hand side (LHS) of constraint attributes, if the LHS attribute values of two tuples are identical, these two tuples are probably duplicate.



(a) Feature weights (Abt - Buy)

(b) Feature weights (Amazon-Google)

Figure 3.16: Feature weights

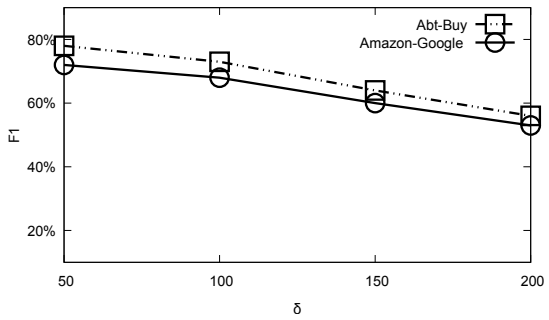


Figure 3.17: Δ Sensitivity

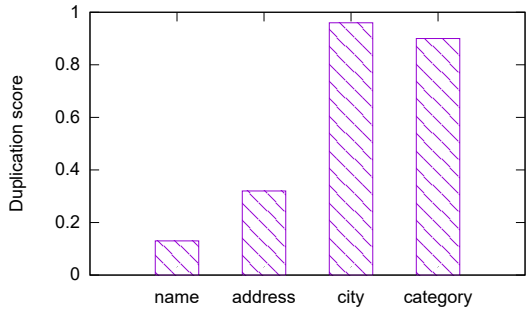


Figure 3.18: Dup. scores (Restaurant)

3.9.7 Sensitivity to Δ

We vary the MFD tolerance parameter Δ to evaluate its sensitivity on Abt-Buy and Amazon-Google datasets. Figure 3.17 shows the accuracy F1 as we vary Δ in MFD [product name] $\xrightarrow{\Delta}$ [price]. The results show that as Δ increases, F1 values slowly decrease on both datasets. The Δ parameter influences f_4 directly and is considered in f_3. The decrease in F1 accuracy is due to the increased number of false positive tuples that are included due to larger Δ values. Despite this decrease, our framework relies on the remaining features to compensate for this F1 loss.

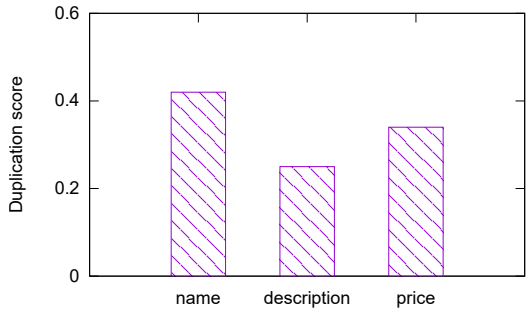


Figure 3.19: Dup. scores (Abt-Buy)

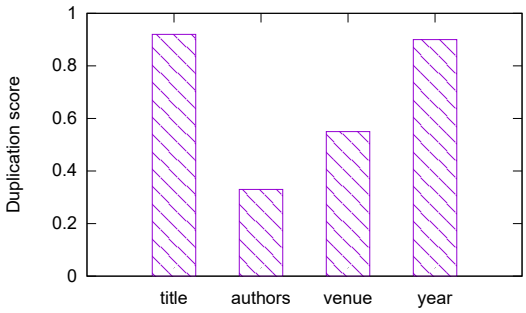


Figure 3.20: Dup. scores (DBLP-ACM)

3.9.8 Accuracy of Duplication Scores

We empirically evaluate the accuracy of our proposed duplication score using the Restaurants, Abt-Buy and DBLP-ACM datasets, by computing *dup(A)* for each column in these datasets separately. Figure 3.18 shows the results for each column of Restaurant dataset, namely, *name, address, city, category*. We observe that the scores for *name* and *address* are lower, indicating the presence of more distinct values in these columns. Upon observation, we recognize that restaurant names are primarily unique, and their addresses are also distinct but share common terms such as street name (e.g., *2450 Broadway St*, *550 Broadway St*). In contrast, we obtain larger duplication scores for attributes *city* and *category*, as these are duplicated across restaurants. For example, multiple restaurants belong to a city, and the category describes regional cuisine, such as *Chinese*, *American*, or *Italian*, where multiple restaurants also share the same value.

Similarly, Figure 3.19 and Figure 3.20 shows the duplication scores for each column of Abt-Buy and DBLP-ACM respectively. Since the tuples collected from Abt.com do not contain the *manufacturer* column, we only include the other three attributes in Figure 3.19. We observe that scores of all three attributes in the Abt-Buy dataset

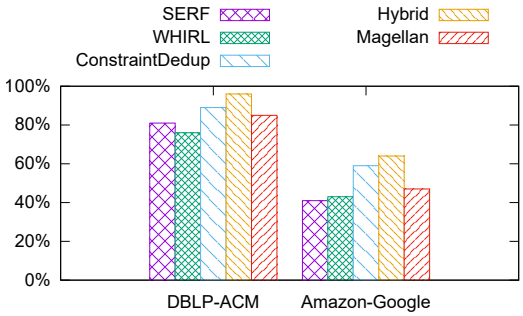


Figure 3.21: Comparative Precision

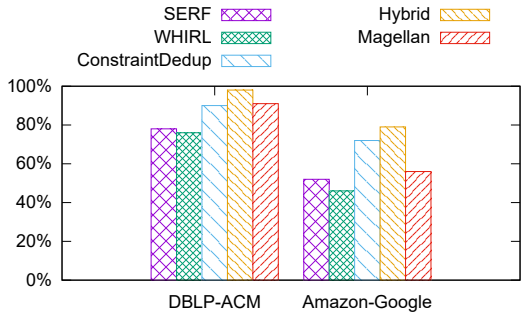


Figure 3.22: Comparative Recall

are relatively low, especially the score of *description* is only 0.23. This is caused by long sentences that exist in the *description* attribute, resulting in lower accuracy. In contrast, we obtain large duplication scores for attributes *title* and *year* in the DBLP-ACM dataset.

3.9.9 Comparative Baseline Evaluation

We compare our approach, ConstraintDeDup against traditional (non-learning) approaches WHIRL[33] and SERF [35], and recent ER learning approaches Magellan [40] and Hybrid [41].

Figure 3.21 and Figure 3.22 show the comparative precision and recall, respectively, of ConstraintDedup against the four baseline techniques (WHIRL, SERF, Hybrid and Magellan) using the DBLP-ACM and Amazon-Google datasets. We observe that all approaches have relative low precision and recall on the Amazon-Google product dataset due to its unstructured string values in the product attribute.

Magellan generates the string similarity-based features from tuple attributes, and leverages traditional machine learning algorithms to train these features. Magellan relies on string similarity to capture the features, which is not suitable for synonyms

with small string similarity. Our work uses word-embeddings to represent the word vector and leverages constraints to capture the relationship between attributes, and model the matching process as a classification problem. The Hybrid approach also treats the ER as a classification problem, which is in the same spirit as ours. The difference is that the Hybrid approach treats a tuple as a long string, and uses a Bidirectional Recurrent neural network (Bi-RNN) with the attention model to represent the embedding vectors. Therefore, Hybrid needs much more training time than our model due to the complexity of neural networks although it can obtain better results.

On average, ConstraintDedup achieves 18% gain on precision and 21% gain on recall over traditional approaches WHIRL and SERF. As our approach considers both the semantic meaning of attribute values (through word-embeddings) and the relations among the attributes (through constraint-based features), it outperforms the string similarity-based approaches WHIRL and SERF.

Our approach performs comparably with Magellan, but the gains are quite different on two datasets. We can see that the gains achieved by our approach on DBLP-ACM (4%) are relatively smaller than the gain on Amazon-Google (23%). This is because the product tuples in the Amazon-Google dataset contain many synonyms, which are semantically similar with large string distances. Magellan relies on the string similarities among attribute values to generate features for classification, and does not consider semantic similarity among strings, thereby leading to lower accuracy values over synonyms.

3.9.10 Runtime Evaluation

We evaluate the runtime of our approach against existing techniques. The runtime is the total overhead of each module in our framework, which includes record blocking, feature extraction, integration, training and classification. Since we use the pre-trained word-embeddings from [53, 54] , our runtime does not include the training time of the word-embeddings. Table 3.10 shows the runtime of each approach on DBLP-ACM and Amazon-Google datasets. The runtime of Magellan is the lowest among all approaches. The runtime of ConstraintDedup is only $0.12h$ more than Magellan (on average), but achieves 16% higher precision and recall than Magellan on the DBLP-ACM and Amazon-Google datasets.

The Hybrid approach gains +7% in precision and +8.5% in recall than our ConstraintDedup, but our ConstraintDedup takes much less runtime than Hybrid, as shown in Table 3.10. Specifically, ConstraintDedup achieves -8% on accuracy but leads to an average -60% runtime decrease against Hybrid. This is because the Hybrid model relies on the complex structure of neural networks and the attention mechanism to explore the implicit semantic features of the dataset to achieve high accuracy, which also requires a high time complexity due to its complex network structure. The feature vectors defined in our approach ConstraintDedup are easy to compute since most of the features are based on frequency and number of tuples. Moreover, the learning model we used in our framework are basic machine learning models (such as SVM, Logistic Regression, Random Forest, Naive Bayes), so the training time of our model is far less than Hybrid.

	SERF	WHIRL	ConstraintDedup	Hybrid	Magellan
DBLP-ACM	1.31h	1.45h	0.58h	2.30h	0.45h
Amazon-Google	1.18h	1.26h	0.45h	2.13h	0.32h

Table 3.10: Runtime comparison

3.10 Conclusion

In this chapter, we treat ER as a classification problem and present a framework that combines constraint and non-constraint attribute features to identify duplicates. In order to capture the relationship between attributes, we leverage data integrity constraints defined in a relation, and propose a set of statistical features to capture attribute and constraint properties to use as input to a classifier. These features provide summaries for researchers and engineers as input feature vectors in machine learning techniques. We also propose a weighted frequency metric, which differentiates the relative weight of terms in a record during the matching process. This weighted frequency approach can easily extend to distance-based similarity methods to help them to place greater emphasis on terms that are better indicators of duplicates, and discounting commonly occurring terms in the data that do not serve as good differentiators. We also proposed metrics that measure the level of duplication for a specific attribute value, and for an attribute domain. We conduct an extensive evaluation of our comparative accuracy against four existing baseline solutions. We showed that our techniques achieve improved runtime performance against existing learning-based models, while achieving comparative accuracy. As next steps, we will experiment with more use cases to refine our duplication metrics to handle greater heterogeneity of data types, and extend our features to include a broader set of data

integrity constraints. We will also investigate how to extend our framework to minimize training effort for similar datasets. The challenge is how to determine whether two datasets are similar. Initially, if two datasets have the same schema, the same predefined constraints and the distribution of their attribute values are close to each other, then we may consider they are similar datasets, and train our model on one dataset and apply it to the other one. If two datasets are partially similar, we can train a basic model on the common attributes to save training time. Specifically, we can build upon this basic model by incrementally applying the constraint features that are unique to each dataset while still leveraging their shared properties to save computation.

In the next chapter, we now turn our attention to the problem of data privacy and data cleaning. This problem is important due to the rise of concern about data privacy, and the need to protect sensitive information and minimize information disclosure. There has been limited work coupling data privacy and data cleaning, and we introduce our framework to bridge these two data management challenges in the next chapter.

Chapter 4

Privacy-Preserving Data Cleaning

4.1 Introduction

Data cleaning is a pervasive problem motivated by the fact that real data is rarely error free. Organizations continue to be hindered by poor data quality as they wrangle with their data to extract value. Recent studies estimate that up to 80% of the data analysis pipeline is consumed by data preparation tasks such as data cleaning. A wealth of data cleaning solutions have been proposed to reduce this effort: constraint based cleaning that use dependencies as a benchmark to repair data values such that the data and dependencies are consistent [60], statistical based cleaning, which propose updates to the data according to expected statistical distributions [61], and leveraging master data as a source of ground truth [14].

Recent advances in networking and cloud infrastructure have motivated a new computing paradigm called *Database-as-a-Service* that lowers the cost and increases access to a suite of data management services. A *service provider* provides the necessary hardware and software platforms to support a variety of data management

tasks to a *client*. Companies such as Amazon, Microsoft, and Google each provide storage platforms, accelerated computational capacity, and advanced data analytics services. However, the adoption of data cleaning services has been limited due to privacy restrictions that limit data sharing. Recent data cleaning efforts that use a curated, master data source share a similar service model to provide high quality data cleaning services to a client with inconsistent data [14, 15]. However, these techniques largely assume the master data is widely available, without differentiating information sensitivity among the attribute values.

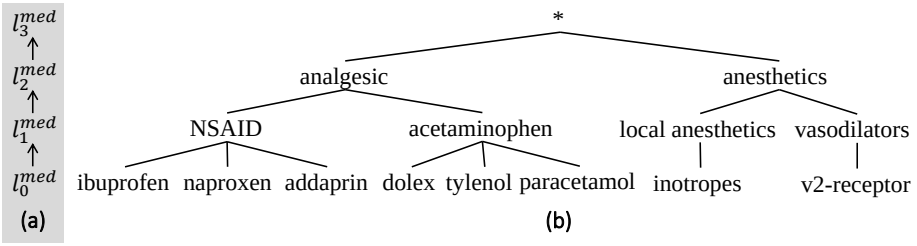


Figure 4.1: (a) $\mathsf{DGH}^{\mathsf{med}}$ and (b) $\mathsf{VGH}^{\mathsf{med}}$.

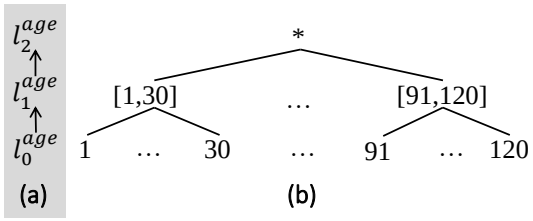


Figure 4.2: (a) $\mathsf{DGH}^{\mathsf{age}}$ and (b) $\mathsf{VGH}^{\mathsf{age}}$.

Example 4.1.1. A data broker gathers longitudinal data from hospital and doctors' records, prescription and insurance claims. Aggregating and curating these disparate datasets lead to a valuable commodity that clients are willing to pay for gleaned insights, and to enrich and clean their individual databases. Table 4.1 shows the curated data containing patient demographic, diagnosis and medication information.

ID	GEN	AGE	ZIP	DIAG	MED
m_1	male	51	P0T2T0	osteoarthritis	ibuprofen
m_2	female	45	P2Y9L8	tendinitis	addaprin
m_3	female	32	P8R2S8	migraine	naproxen
m_4	female	67	V8D1S3	ulcer	tylenol
m_5	male	61	V1A4G1	migraine	dolex
m_6	male	79	V5H1K9	osteoarthritis	ibuprofen

Table 4.1: Curated medical records (R_{SP})

The schema consists of patient gender (GEN), age (AGE), zip code (ZIP), diagnosed illness (DIAG), and prescribed medication (MED).

A client such as a pharmaceutical firm, owns a stale and inconsistent subset of this data as shown in Table 4.2. For example, given a functional dependency (FD) $\varphi : [\mathsf{GEN}, \mathsf{DIAG}] \rightarrow [\mathsf{MED}]$ on Table 4.2, it states that a person's gender and diagnosed condition determine a prescribed medication. That is, for any two tuples in Table 4.2, if they share equal values in the [GEN, DIAG] attributes, then they should also have equal values in the MED attribute. We see that tuples t_1 - t_5 falsify φ. Error cells that violate φ are highlighted in light and dark gray, and values in bold are inaccurate according to Table 4.1.

If the client wishes to clean her data with the help of a data cleaning service provider (i.e., the data broker and its data), she must first match the inconsistent records in Table 4.2 against the curated records in Table 4.1. This generates possible fixes (also known as repairs) to the data in Table 4.2. The preferred repair is to update $t_2[\mathsf{MED}]$, $t_3[\mathsf{MED}]$ to *ibuprofen* (from m_6), and $t_4[\mathsf{DIAG}]$ = *migraine* (from

ID	GEN	AGE	DIAG	MED
t_1	male	51	osteoarthritis	ibuprofen
t_2	male	79	osteoarthritis	**intropes**
t_3	male	45	osteoarthritis	**addaprin**
t_4	female	32	**ulcer**	naproxen
t_5	female	67	ulcer	tylenol
t_6	male	61	migrane	dolex
t_7	female	32	pylorospasm	appaprtin
t_8	male	37	hypertension	dolex

Table 4.2: Dirty client records w.r.t. φ.

ID	GEN	AGE	ZIP	DIAG	MED
g_1	*	[31,60]	P*	osteoarthritis	ibuprofen
g_2	*	[31,60]	P*	tendinitis	addaprin
g_3	*	[31,60]	P*	migraine	naproxen
g_4	*	[61,90]	V*	ulcer	tylenol
g_5	*	[61,90]	V*	migraine	dolex
g_6	*	[61,90]	V*	osteoarthritis	ibuprofen

Table 4.3: Public table.

m_3). However, this repair may disclose sensitive patient information about diagnosed illness and medication from the service provider. It may be preferable to disclose a more general value that is semantically similar to the true value to protect individual privacy. For example, instead of disclosing the medication *ibuprofen*, the service provider discloses *Non-steroid anti-inflammatory drug (NSAID)*, which is the family of drugs containing *ibuprofen*. In this chapter, we explore how to compute such generalized repairs in an interactive model between a service provider and a client to protect sensitive data and to improve accuracy and consistency in client data.

State-of-the-Art. Existing work in data privacy and data cleaning have been limited to imputation of missing values using decision trees [62], or information-theoretic techniques [63], and studying trade-offs between privacy bounds and query accuracy over differentially private relations [64]. In the *data cleaning-as-a-service* setting, which we consider in this chapter, using differential privacy poses the following limitations: (i) differential privacy provides guarantees assuming a limited number of interactions between the client and service provider; (ii) queries are limited to aggregation queries; and (iii) data randomization decreases data utility.

Given the above limitations, we explore the use of *Privacy Preserving Data Publishing* (PPDP) techniques, that even though do not provide the same provable privacy guarantees as differential privacy, do restrict the disclosure of sensitive values without limiting the types of queries nor the number of interactions between the client and service provider. PPDP models prevent re-identification and break attribute linkages in a published dataset by hiding an individual record among a group of other individuals. This is done by removing identifiers and generalizing *quasi-identifier* (QI) attributes (e.g., GEN, AGE, ZIP) that together can re-identify an individual.

Well-known PPDP methods such as *k-anonymity* require the group size to be at least k such that an individual cannot be identified from $k-1$ other individuals [6, 65]. Extensions include *(X, Y)-anonymity* that break the linkage between the set X of QI attributes, and the set Y of sensitive attributes by requiring at least k distinct sensitive values for each unique X [66]. For example, Table 4.3 is k-anonymous for $k = 3$ by generalizing values in the QI attributes. It is also (X,Y)-anonymous for sensitive attribute MED for $k = 3$ since there are three distinct medications for each value in the QI attributes X, e.g., values $(*, [31, 60], P*)$ of X co-occur with *ibuprofen*, *addaprin*, and *naproxen* of Y.

To apply PPDP models in data cleaning, we must also address the highly contextualized nature of data cleaning, where domain expertise is often needed to interpret the data to achieve correct results. It is vital to incorporate these domain semantics during the cleaning process, and into a privacy model during privacy preservation. Unfortunately existing PPDP models only consider syntactic forms of privacy via generalization and suppression of values, largely ignoring the data semantics. For example, upon closer inspection of Table 4.3, the values in the QI attributes in records $g_1 - g_3$ are associated with a single medication, since *ibuprofen*, *addaprin*, and *naproxen* are all references to the same medication marketed under different brand names. We prevent this semantic privacy loss by defining an extension of (X,Y)-anonymity that incorporates a *generalization hierarchy*, which capture the semantics of an attribute domain.[1] In Table 4.3, the medications in $g_1 - g_3$ are modeled as synonyms in such a hierarchy.

Example 4.1.2. To clean Table 4.2, a client requests the correct value(s) in tuples

[1] k-anonymity and its extensions such as l-diversity and t-closeness also do not consider the underlying data semantics.

$t_1 - t_3$ from the data cleaning service provider. The service provider returns *ibuprofen* to the client to update $t_2[\mathsf{MED}]$ and $t_3[\mathsf{MED}]$ to $m_6[\mathsf{MED}] = \textit{ibuprofen}$. However, if disclosing the specific *ibuprofen* medication violates the requirements of the underlying privacy model, then a revised solution to disclose a less informative, generalized value, such as *analgesic* is preferred.

Technical Challenges.

1. Data cleaning with functional dependencies (FDs), and record matching (between the service provider and client) is an NP-complete problem. In addition, it is approximation-hard, i.e., the problem cannot be approximately solved with a polynomial-time algorithm and a constant approximation ratio [14]. We extend this data cleaning problem with privacy restrictions on the service provider, making it as hard as the initial problem. We analyze the complexity and propose a solution realizable in practice.

2. Given a generalization hierarchy as shown in Figures 4.1 and 4.2, proposed repairs from a service provider may contain general values that subsume a set of specific values at the leaf level. In such cases, we study a new (repair) semantics for consistency with respect to (w.r.t.) an FD involving general values.

3. There exists a space of possible data instances that can be published by the service provider to facilitate data cleaning with the client, while protecting individual entities and sensitive values. Quantifying these possible instances and developing an interaction model between the service provider and client is our third challenge.

4. Resolving inconsistencies w.r.t. a set of FDs requires traversing the space of possible fixes (which may now include general values). By proposing generalized values as repair values, we lose specificity in the client data instance. We study the trade-off between high confidence, generalized repair values versus lower confidence, specific repairs.

Our Approach and Contributions. We build upon our earlier work that introduces *PACAS*, a *P*rivacy-*A*ware data *C*leaning-*A*s-a-*S*ervice framework that facilitates data cleaning between a client and a service provider [11]. The interaction is done via a data pricing scheme where the service provider charges the client for each disclosed value, according to its adherence to the privacy model. PACAS includes a new privacy model that extends (X,Y)-anonymity to consider the data semantics, while providing stronger privacy protection than existing PPDP methods. We present a data cleaning algorithm that resolves errors (FD violations) by updating them to values that are semantically equivalent to their true value(s) in the service provider data. We make the following contributions:

1. We first present PACAS, a privacy-preserving, data cleaning framework that identifies errors w.r.t. a set of FDs in client data, and allows the client to purchase clean, curated data from a service provider.

2. We introduce *generalized queries* to allow a client to request data values from the service provider at different levels of generalization.

3. We extend the repair semantics of our previous model [11], and introduce *generalized repairs* that update values in the client instance to generalized values.

We re-define the notion of consistency between a relational instance (with generalized values) and a set of FDs. We propose an entropy-based measure that quantifies the semantic distance between two (generalized) values to evaluate the utility of repair candidates.

4. We introduce (X,Y,L)-anonymity that considers the data semantics for an attribute domain via generalization hierarchies w.r.t. a level L. We apply (X,Y,L)-anonymity at the service provider to protect sensitive attributes. We present *SafePrice*, a data pricing algorithm that allows the service provider to sell data to the client, while preserving (X,Y,L)-anonymity over sensitive data.

5. We present the *SafeClean* algorithm that resolves inconsistencies w.r.t. a set of FDs by using external data purchased from a service provider. *SafeClean* proposes repairs to the data by balancing its data privacy requirements against satisfying query purchase requests for its data at a computed price. Given a cleaning budget, we present a new budget allocation algorithm that improves upon previous, fixed allocations [11], to consider allocations according to the number of errors in which a database value participates. We analyze the runtime complexity showing the efficiency of its components.

6. We evaluate the effectiveness of *SafePrice* and *SafeClean* over real data showing that they achieve an average +46% in F1 repair accuracy over existing solutions, and conceal +32% more sensitive values than (X,Y)-anonymity. We also show that our approach achieves improved repair accuracy across varying budget parameter and maintains 70-80% repair accuracy for error rates up to 15%.

In Section 4.2, we present the related work. In Section 4.3, we provide notation and preliminaries, followed by definitions for a generalized relation and consistency of a generalized relation w.r.t. a set of FDs in Section 4.4. We present our problem definition, and introduce our system *PACAS* in Section 4.5. We define generalized queries, and how to price these queries in Section 4.6, and describe our data cleaning algorithm that considers generalized values in Section 4.7. We present our experimental results in Section 4.8, and conclude in Section 4.9.

4.2 Related Work

Our work finds relation to data privacy, data cleaning, and data pricing.

Data Privacy and Data Cleaning. We extend the PACAS framework introduced in [11], which focuses on repairs involving ground values. In this work, we extend the space of repairs to consider generalized values to facilitate anonymization, and propose a new entropy-based distance metric to quantify the data cleaning utility by measuring the information loss. We have also presented a new definition of consistency given generalized repair values, and an extended repair algorithm that preferentially allocates larger budgets to the dirtiest cells leading to the greatest reduction in the number of errors. Our experiments show the influence of generalized repairs along varying parameters towards improved efficiency and repair accuracy.

PrivateClean is a framework for data cleaning and approximate query processing on locally differentially private relations that combines data cleaning and local differential privacy [64]. Based on generalized randomized response, PrivateClean presents user-defined data cleaning operations in the form of extraction, merging, and transformation. While generic user-defined operations are given, PrivateClean relies on

the user to provide the specific cleaning semantics. In addition, as shown in our evaluation, error detection and cleaning over randomized data is difficult, and decreases data utility. Apex is another framework which focuses on the accuracy bound of the differential private queries in data exploration and cleaning [67]. It allows analysts to directly specify accuracy bounds on the query and converts the query with accuracy bound to a query with privacy guarantee. It works well on another data cleaning task(entity resolution), which is different from the data consistency cleaning task in this chapter. *SafeClean* provides a dependency based data cleaning solution on data consistency that can be realized in practice with consideration for sensitive data while maintaining good accuracy rates.

Dependency Based Cleaning. Declarative approaches to data cleaning (cf. [68] for a survey) have focused on achieving consistency w.r.t. a set of dependencies such as FDs and inclusion dependencies and their extensions. Data quality semantics are declaratively specified with the dependencies, and data values that violate the dependencies are identified as errors, and repaired [15, 16, 17, 20, 21]. There are a wide range of cleaning algorithms, based on user/expert feedback [22, 23, 24], master database [14, 15], knowledge bases or crowdsourcing [25], probabilitic inference [18, 19]. Our repair algorithm builds upon these existing techniques to include data disclosure requirements of sensitive data and suggesting generalized values as repair candidates.

Data Privacy. PPDP uses generalization and suppression to limit data disclosure of sensitive values [6, 65]. The generalization problem has been shown to be intractable, where optimization, and approximation algorithms have been proposed (cf. [66] for a survey). Extensions have proposed tighter restrictions to the baseline

k-anonymity model to protect against similarity and skewness attacks by considering the distribution of the sensitive attributes in the overall population in the table [66]. Our extensions to (X,Y)-anonymity to include semantics via the VGH can be applied to existing PPDP techniques to semantically enrich the generalization process such that semantically equivalent values are not inadvertently disclosed.

In differential privacy, the removal, addition or replacement of a single record in a database should not significantly impact the outcome of any statistical analysis over the database [69]. An underlying assumption requires that the service provider know in advance the set of queries over the released data. This assumption does not hold for the interactive, service-based data cleaning setting considered in this chapter. *PACAS* aims to address this limitation.

Data Pricing. We use the framework by Deep and Koutris that provides a scalable, arbitrage-free pricing for SQL queries over relational databases [70]. Recent work has considered pricing functions to include additional properties such as being *reasonable* (differentiated query pricing based on differentiated answers), and *predictable*, *non-disclosive* (the inability to infer unpaid query answers from query prices) and *regret-free* (asking a sequence of queries during multiple interactions is no more costly than asking at once) [71]. We are investigating extensions of *SafePrice* to include some of these properties.

4.3 Preliminaries

We present necessary notations and definitions.

4.3.1 Matching Dependencies

A *matching dependency* (MD) ϕ over two relations R and R' with schemata $\mathcal{R} = \{A_1, A_2, ...\}$ and $\mathcal{R}' = \{A'_1, A'_2, ...\}$ is an expression of the following form:

$$\bigwedge_{i\in[1,n]} R[X_i] \approx R'[X'_i] \rightarrow R[Y] = R'[Y'], \tag{4.1}$$

where (X_i, X'_i) and (Y, Y') are comparable pairs of attributes in R and R'. The MD ϕ states that for a pair of tuples (t, t') with $t \in R$ and $t' \in R'$, if $t[X'_i]$ values are similar to values $t'[X'_i]$ according to the similarity function $\approx$, the values of $t[Y]$ and $t'[Y']$ are identical [15].

4.3.2 (X,Y)-Anonymity

Definition 4.3.1 ((X,Y)-anonymity)**.** *A table R with schema $\mathcal{R}$ and attributes $X, Y \subseteq \mathcal{R}$ is* (X,Y)-anonymous *with value k if for every $t \in R$, there are at least k values in $Q^t(R)$, $|Q^t(R)| \geq k$, where $Q^t(R) = \Pi_Y(\sigma_{X=t[X]}(R))$.*

Example 4.3.1. For $X = \{\mathsf{GEN}, \mathsf{AGE}, \mathsf{ZIP}\}$, $Y = \{\mathsf{MED}\}$ with $k = 3$, Table 4.3 is (X,Y)-anonymous since each $g_i \in R$, $Q^{g_i}(R)$ is either $\{$*ibuprofen*, *addaprin*, *naproxen*$\}$ or $\{$*tylenol*, *dolex*, *ibuprofen*$\}$. Table 4.3 is not (X,Y)-anonymous with $X = \{\mathsf{DIAG}\}$ and $Y = \{\mathsf{MED}\}$, since for g_1, $Q^{g_1}(R) = \{$*ibuprofen*$\}$ with size $1 \leq k$.

The (X,Y)-anonymity model extends k-anonymity; if Y is a key for R and X, Y are QI and sensitive attributes, respectively, then (X,Y)-anonymity reduces to k-anonymity.

Symbol	Description
$\mathsf{Dom}^A, \mathsf{dom}^A$	domain of attribute A
$\mathsf{dom}^A(l)$	sub-domain of attribute A in level l
DGH^A,VGH^A	domain and value generalization hierarchies
$\leq$	generalization relation for levels
$\preceq$	generalization relation for values and tuples
Q, G	Simple query and Generalized query (GQ)
l, L	level and sequence of levels
$\mathcal{S}, \mathcal{C}^Q$	support set, and conflict set
$\mathcal{B}, B_i$	total and the budget for the i-th iteration
l_{max}	generalization level
$C_{err} = \{e_1, e_2, ...\}$	set of candidate error cells for repair
c, e	a database cell and an error cell
δ	distance function for values, tuples, tables

Table 4.4: Summary of notation and symbols.

The l-diversity privacy model extends k-anonymity with a stronger restriction over the X-groups [72]. A relation is considered l-diverse if each X-group contains at least l "well-represented" values in the sensitive attributes Y. Well-representation is normally defined according to the application semantics, e.g., *entropy l-diversity* requires the entropy of sensitive values in each X-group to satisfy a given threshold [72]. When well-representation requires l sensitive values in each Y attribute, (X,Y)-anonymity reduces to l-diversity.

4.3.3 Generalization

Generalization replaces values in a private table with less specific, but semantically consistent values according to a generalization hierarchy. To generalize attribute A, we assume a set of levels $\mathcal{L}^A = \{l_0^A, ..., l_h^A\}$ and a partial order $\leq^A$, called a *generalization relation* on $\mathcal{L}^A$. Levels l_i^A are assigned with disjoint domain-sets $\mathsf{dom}(l_i^A)$. In $\leq^A$, each level has at most one parent. The domain-set $\mathsf{dom}(l_n^A)$ is the maximal domain set and it is a singleton, and $\mathsf{dom}(l_0^A)$ is the *ground domain set*. The definition of $\leq^A$ implies the existence of a totally ordered hierarchy, called the *domain generalization hierarchy*, DGH^A. The domain set $\mathsf{dom}(l_i^A)$ generalizes $\mathsf{dom}(l_j^A)$ iff $l_j^A \leq l_i^A$. We use h^A to refer to the number of levels in DGH^A. Figures 4.1(a) and 4.2(a) show the DGH s for the medication and age attributes, resp. A *value generalization relationship* for A, is a partial order $\preceq^A$ on $\mathsf{Dom}^A = \bigcup \mathsf{dom}(l_i^A)$. It specifies a *value generalization hierarchy*, VGH^A, that is a tree whose leaves are values of the ground domain-set $\mathsf{dom}(l_0^A)$ and whose root is the single value in the maximal domain-set $\mathsf{dom}(l_n^A)$ in DGH^A. For two values v and v' in Dom^A, $v' \preceq^A v$ means v' is more specific than v according to the VGH. We use $\preceq$ rather than $\preceq^A$ when the attribute is clear from the

context. The VGH for the MED and AGE attributes are shown in Figures 4.1(b) and 4.2(b), respectively. According to VGH of MED, *ibuprofen* $\preceq$ *NSAID*.

A value is *ground* if there is no value more specific than it, and it is *general* if it is not ground. In Figure 4.1, *ibuprofen* is ground and *analgesic* is general. For a value v, its base denoted by $\mathsf{base}(v)$ is the set of ground values u such that $u \preceq v$. We use $0 \leq \mathsf{level}(v) \leq h^A$ to refer to the level of v according to VGH^A.

A *general relation* (table) is a relation with some general values and a ground relation has only ground values. A *general database* is a database with some general relations and a ground database has only ground relations. The generalization relation $\preceq^A$ trivially extends to tuples (denoted by $\sqsubseteq$). We give an extension of the generalization relation to general relations and databases in Section 4.4.

The generalization hierarchies (DGH and VGH) are either created by the data owners with help from domain experts or generated automatically based on data characteristics [73]. The automatic generation of hierarchies for categorical attributes [74] and numerical attributes [75, 76, 77] apply techniques such as histogram construction, binning, numeric clustering, and entropy-based discretization [73].

4.3.4 Data Pricing

High quality data is a valuable commodity that has lead to increased purchasing and selling of data online. Data market services such as AggData [78], Infochimps [79] and Kaggle [80] have become popular vendors in recent years and query pricing has been proposed as a fine-grained and user-centric technique to support the exchange and marketing of data [71].

Given a database instance D and query Q, a *pricing function* returns a non-negative real number representing the price to answer Q [71]. A pricing function should have the desirable property of being *arbitrage-free* [71]. Arbitrage is defined using the concept of *query determinacy* [81]. Intuitively, Q determines Q' if for every database D, the answers to Q' over D can be computed from the answers to Q over D. Arbitrage occurs when the price of Q is less than that of Q', which means someone interested in purchasing Q' can purchase the cheaper Q instead, and compute the answer to Q' from Q. For example, $Q(R) = \Pi_Y(R)$ determines $Q'(R) = \Pi_Y(\sigma_{Y<10}(R))$ because the user can apply $Y < 10$ over $Q(R)$ to obtain $Q'(R)$. Arbitrage occurs if Q is cheaper than Q' which means a user looking for $Q'(R)$ can buy Q and compute answers to Q'. An arbitrage-free pricing function denies any form of arbitrage and returns consistent pricing without inadvertent data leakage [71]. A pricing scheme should also be *history-aware* [71]. A user should not be charged multiple times for the same data purchased through different queries. In such a pricing scheme, the data seller must track the history of queries purchased by each buyer and price future queries according to historical data. In Section 4.6.2, we present an arbitrage-free and history-aware data pricing scheme that extends the pricing model in [70].

4.4 Generalized Relations

Using general values as repair values in data cleaning requires us to re-consider the definition of consistency between a relational instance and a set of FDs. In this section, we introduce an entropy-based measure that allows us to quantify the utility of a generalized value as a repair value, and present a revised definition of consistency that has not been considered in past work [11].

4.4.1 Measuring Semantic Distance

By replacing a value v' in a relation R with a *generalized* value v, there is necessarily some information loss. We present an entropy-based penalty function that quantifies this loss [82]. We then use this measure as a basis to define a distance function between two general values.

Definition 4.4.1 (Entropy-based Penalty [82])**.** Consider an attribute A in a ground relation R. Let X_A be a random variable to randomly select a value from attribute A in R. Let VGH^A be the value generalization hierarchy of the attribute A (the values of A in R are ground values from VGH^A). The entropy-based penalty of a value v in VGH^A denoted by $E(v)$ is defined as follows:

$$E(v) = P(X_A \in \mathit{base}(v)) \times H(X_A | X_A \in \mathit{base}(v)),$$

where $P(X_A \in \mathit{base}(v))$ is the probability that the value of X_A is a ground value and a descendant of v, $X_A \in \mathit{base}(v)$). The value $H(X_A | X_A \in \mathit{base}(v))$ is the entropy of X_A conditional to $X_A \in \mathit{base}(v)$.

Intuitively, the entropy $H(X_A | X_A \in \mathit{base}(v))$ measures the uncertainty of using the general value v. $E(v)$ measures the information loss of replacing values in $\mathit{base}(v)$ with v. Note that $E(v) = 0$ if v is ground because $H(X_A | X_A \in \mathit{base}(v)) = 0$, and $E(v)$ is maximum if v is the root value $*$ in VGH^A. $E(v)$ is monotonic whereby if $v \preceq v'$, then $E(v) \leq E(v')$. Note that the conditional entropy $H(X_A | X_A \in \mathit{base}(v))$ is not a monotonic measure [82].

Example 4.4.1. Consider the general value $v = g_1[\mathsf{AGE}]$ =[31,60] in Table 4.1. Three ground values *51*, *45* and *32* in $\mathit{base}(v)$ appear in Table 4.1, which has total

6 records, so $P(X_{\mathsf{AGE}} \in base(v)) = \frac{3}{6} = \frac{1}{2}$. According to Table 4.1, the conditional entropy is $H(X_A|X_{\mathsf{AGE}} \in base(v)) = 3 \times (-\frac{1}{3} \times \log \frac{1}{3}) = 1.58$ because *51*, *45* and *32* each appear exactly once in the table. Therefore, the entropy-based penalty of v is $E(v) = \frac{1}{2} \times 1.58 = 0.79$.

Definition 4.4.2 (Semantic Distance Function, δ)**.** The semantic distance between v and v' in VGH^A is defined as follows:

$$\delta(v, v') = \begin{cases} |E(v') - E(v)| & \text{if } v \preceq v' \text{ or } v' \preceq v \\ \delta(v, a) + \delta(a, v') & \text{otherwise} \end{cases}$$

in which $a = lca(v, v')$ is the least common ancestor of v and v'.

Intuitively, if v is a descendant of v' or vice versa, i.e. $v \preceq v'$ or $v' \preceq v$, their distance is the the difference between their entropy-based penalties, i.e., $|E(v')-E(v)|$. This is the information loss incurred by replacing a more informative child value v with its ancestor v' when $v \preceq v'$. If v and v' do not align along the same branch in the VGH, i.e. $v \npreceq v'$ and $v' \npreceq v$, $\delta(v, v')$ is the total distance between v and v' as we travel through their least common ancestor a, i.e. $\delta(v, a) + \delta(a, v')$.

Example 4.4.2. (Ex. 4.4.1 continued) According to Definition 4.4.2, $\delta([31, 60], 51) = |E([31, 60])-E(51)| = |0.79-0| = 0.79$ because $51 \preceq [31, 60]$. Similarly, $\delta([31, 60], 45) = 0.79$. Also, $\delta(45, 51) = \delta(45, [31, 60]) + \delta([31, 60], 51) = 1.58$ because 45 and 51 do not belong to the same branch, and $[31, 60]$ is their least common ancestor.

The $\delta(v'v)$ distance captures the semantic closeness between values in the VGH. We extend the definition of δ to tuples by summing the distances between corresponding

values in the two tuples. The δ function naturally extends to sets of tuples and relations. We use the $\delta(v'v)$ distance measure to define repair error in our evaluation in Section 4.8.

4.4.2 Consistency in Generalized Relations

A (generalized) relation R may contain generalized values that are syntactically equal but semantically different. For example, $g_1[\mathsf{AGE}]$ and $g_2[\mathsf{AGE}]$ in Table 4.3 are syntactically equal (containing [31,60]), but their true values in Table 4.2 are different, $t_1[\mathsf{AGE}] = 51$ and $t_1[\mathsf{AGE}] = 79$, respectively. The consistency between traditional FDs and a relation R is based on syntactic equality between values. We present a revised definition of consistency with the presence of generalized values in R.

Given a generalized relation R' with schema $\mathcal{R}$, and FD $\varphi : A \rightarrow B$ with $A, B \in \mathcal{R}$, R' satisfies φ ($R' \models \varphi$), if for every pair of tuples $t_1, t_2 \in R'$, if $t_1[A] = t_2[A]$ and $t_1[A], t_2[A]$ are ground values then $t_1[B] \sqsubseteq t_2[B]$ or $t_2[B] \sqsubseteq t_1[B]$. Since a general value v encapsulates a set of distinct ground values in $\mathit{base}(v)$, relying on syntactic equality between two (general) values, $t_1[B]$ and $t_2[B]$, is not required to determine consistency, as they may represent two semantically equivalent entities. Our definition requires that $t_1[B]$ be an ancestor of $t_2[B]$ (or vice-versa). This definition extends to FDs $X \rightarrow Y$ with $X, Y \subseteq \mathcal{R}$, and reduces to the classical FD consistency when R' is ground. Assuming VGHs have fixed size, consistency checking in the presence of generalized values is in quadratic time w.r.t. $|R'|$.

Example 4.4.3. Given FD $\varphi : [\mathsf{GEN}, \mathsf{DIAG}] \rightarrow [\mathsf{MED}]$, according to the classic definition of consistency, t_1 and t_2 in Table 4.2 are inconsistent as $t_1[\mathsf{GEN}, \mathsf{DIAG}] = t_2[\mathsf{GEN}, \mathsf{DIAG}] = \{\mathit{male}, \mathit{osteoarthritis}\}$ but $t_1[\mathsf{MED}] = \mathit{ibuprofen} \neq \mathit{intropes} = t_2[\mathsf{MED}]$.

Under our revised definition of consistency in a generalized relation, if we update $t_2[\mathsf{MED}]$ to *NSAID*, which is the ancestor of *ibuprofen*, then t_1 and t_2 are no longer inconsistent. We have $t_1[\mathsf{GEN}, \mathsf{DIAG}] = t_2[\mathsf{GEN}, \mathsf{DIAG}]$, and $t_1[\mathsf{MED}] = \textit{ibuprofen} \sqsubseteq \textit{NSAID} = t_2[\mathsf{MED}]$ (cf. Fig 4.1 (b) for the VGH of MED). If we update $t_2[\mathsf{MED}]$ to *vasodilators*, which is not the ancestor of $t_1[\mathsf{MED}] = \textit{ibuprofen}$, then t_1 and t_2 remain consistent under the generalized consistency definition.

4.5 PACAS Overview

We formally define our problem, and then give an overview of the PACAS framework.

4.5.1 Problem Statement

Consider a client, CL, and a service provider, SP, with databases D_{CL}, D_{SP} containing single relations R_{CL}, R_{SP}, respectively. Our discussion easily extends to databases with multiples relations. We assume a set of FDs Σ defined over R_{CL} that is falsified. We use FDs as the benchmark for error detection, but our framework is amenable to other error detection methods. The shared generalization hierarchies are generated by the service provider (applying the techniques mentioned in Section 4.3.3). The problem of privacy-preserving data cleaning is twofold, defined separately for CL and SP. We assume a generalization level $l_{\max}$, which indicates the maximum level that values in our repaired database can take from the generalization hierarchy.

Client-side: For every cell $c \in R_{CL}$ with value $c.\mathsf{value}$, let $c.\mathsf{value}^*$ be the corresponding accurate value in R_{SP}. A cell c is considered dirty if $c.\mathsf{value} \neq c.\mathsf{value}^*$. We assume the client can initiate a set of requests $r_1, ..., r_n$ from R_{SP} in which each request r_i

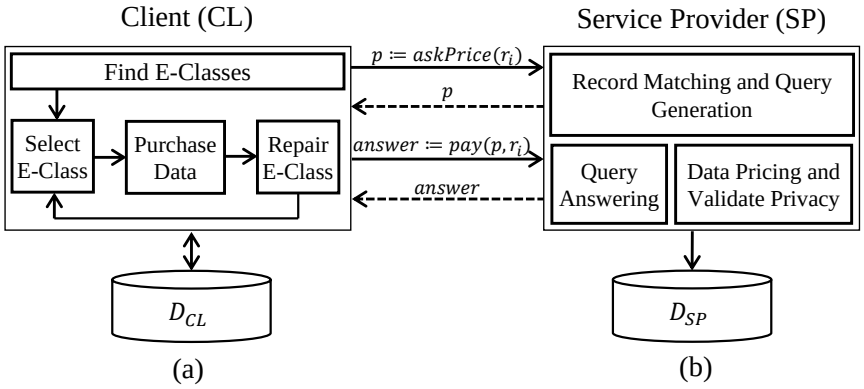


Figure 4.3: Framework overview.

is of the form $r_i = (t, A, l)$, that seeks the clean value of database cell $t[A]$ at level l in R_{SP}. We assume $\sum_i(price(r_i)) \leq \mathcal{B}$ for a fixed cleaning budget $\mathcal{B}$. Let R^*_{CL} be the clean version of R_{CL} where for each cell, $c.\mathsf{value} = c.\mathsf{value}^*$. The problem is to generate a set of requests $r_1, ..., r_n$, where the answers are used to compute a relation R'_{CL} such that: (i) $R'_{CL} \models \Sigma$, (ii) $dist(R'_{CL}, R^*_{CL})$ is minimal, and (iii) for each $c.\mathsf{value}$, its level $l \leq l_{\max}$.

In our implementation, we check consistency $R'_{CL} \models \Sigma$ using the consistency definition in Section 4.4.2, and measure the distance $\delta(R'_{CL}, R^*_{CL})$ using the semantic distance function δ (Defn. 4.4.2).

Service-side: The problem is to compute a pricing function $price(r_i)$ that assigns a price to each request r_i such that R_{SP} preserves (X,Y,L)-anonymity.

4.5.2 Solution Overview

Figure 4.3 shows the PACAS system architecture consisting of two main units that execute functions for *CL* and *SP*. Figure 4.3(a) shows the *CL* unit containing four modules. The first module finds equivalence classes in R_{CL}. An *equivalence class*

(eq) is a set of cells in R_{CL} with equal values in order for R_{CL} to satisfy Σ [60]. The next three modules apply an iterative repair for the eqs. These modules select an eq, purchase accurate value(s) of dirty cell(s) in the class, and then repair the cells using the purchased value. If the repairs contain generalized values, the CL unit must verify consistency of the generalized relation against the defined FDs. The cleaning iterations continue until $\mathcal{B}$ is exhausted or all the eqs are repaired. We preferentially allocated the budget $\mathcal{B}$ to cells in an eq based on the proportion of errors in which the cells (in an eq) participate. Figure 4.3(b) shows the *SP* unit. The *Record Matching* module receives a request $r_i = (t, A, l)$ from *CL*, identifies a matching tuple t' in R_{SP}, and returns $t'[A]$ at the level l according to the VGH. The *Data Pricing and Validate Privacy* module computes prices for these requests, and checks whether answering these requests will violate (X,Y,L)-anonymity in R_{SP}. If so, the request is not safe to be answered. The *Query Answering* module accepts payment for requests, and returns the corresponding answers to *CL*.

4.6 Limiting Disclosure of Sensitive Data

The *SP* must carefully control disclosure of its curated and sensitive data to the *CL*. In this section, we describe how the *SP* services an incoming client request for data, validates whether answering this request is safe (in terms of violating (X,Y,L)-anonymity), and the data pricing mechanism that facilitates this interaction.

4.6.1 Record Matching and Query Generation

Given an incoming *CL* request $r_i = (t, A, l)$, the *SP* must translate r_i to a query that identifies a matching (clean) tuple t' in R_{SP}, and returns $t'[A]$ at level l. To answer each request, the *SP* charges the *CL* a price that is determined by the level l of data disclosure and the adherence of $t'[A]$ to the privacy model. We introduce *generalized queries* (GQs) to access values at different levels of the DGH.

Definition 4.6.1 (Generalized Queries)**.** *A generalized query (GQ) with schema $\mathcal{R}$ is a pair $G = \langle Q, L\rangle$, where Q is an n-ary select-projection-join query over $\mathcal{R}$, and $L = \{l_1, ..., l_n\}$ is a set of levels for each of the n values in Q according to the* DGH*s in $\mathcal{R}$. The set of answers to G over R, denoted as $G(R)$, contains n-ary tuples t with values at levels in $l_i \in \mathsf{L}$, such that $\exists t' \in Q(R)$ and t generalizes t', i.e. $t' \sqsubseteq t$.*

Intuitively, answering a GQ involves finding the answers of $Q(R)$, and then generalizing these values to levels that match L. For a fixed size DGH, the complexity of answering G is the same as answering Q.

Example 4.6.1. Consider GQ with level $L = \{l_0^{\mathsf{GEN}}, l_1^{\mathsf{MED}}\}$ and query $Q(R_{SP}) = \Pi_{\mathsf{GEN},\mathsf{MED}}(\sigma_{\mathsf{DIAG}=migraine}(R_{SP}))$ where R_{SP} is Table 4.1. The answers to $Q(R_{SP})$ is {(*female*, *naproxen*), (*male*, *dolex*)}, which is generalized to {(*female*, *NSAID*), (*male*, *acetaminophen*)} according to L and Figure 4.1.

To translate a request r to a GQ G_r, we assume a schema mapping exists between R_{SP} and R_{CL}, with similarity operators $\approx$ to compare the attribute values. This can be modeled via *matching dependencies* (MDs) in *SP* [15].

Example 4.6.2. To translate r=$(t_2, \mathsf{MED}, l_1^{\mathsf{MED}})$ into G_r, we compare values in the QI attributes, GEN and AGE, and define query request as

$Q_r(R_{SP}) = \Pi_{\mathsf{MED}}(\sigma_{\mathsf{GEN}\approx t_2[\mathsf{GEN}]\wedge\mathsf{AGE}\approx t_2[\mathsf{AGE}]}(R_{SP}))$, $L_r = \{l_1^{\mathsf{MED}}\}$. The query is generated from an assumed MD $R_{CL}[\mathsf{GEN}] \approx R_{SP}[\mathsf{GEN}] \wedge R_{CL}[\mathsf{AGE}] \approx R_{SP}[\mathsf{AGE}] \rightarrow R_{CL}[\mathsf{MED}] = R_{SP}[\mathsf{MED}]$ that says if gender and age of two records in *SP* and *CL* are similar, they refer to the same patient with the same medication.

4.6.2 Enforcing Privacy via Data Pricing

Given a GQ, the *SP* must determine whether it is safe to answer this query, i.e., decide whether disclosing the value requested in r_i violates privacy requirements. We introduce our privacy model (X,Y,L)-anonymity that extends existing PPDP methods to consider the attribute domain semantics, and define the *SafePrice* data pricing algorithm that assigns prices to GQs and guarantees (X,Y,L)-anonymity.

(X,Y,L)-Anonymity. Data randomization used in differential privacy can lead to large data perturbations that render low data utility for data cleaning applications. (X,Y,L)-anonymity extends (X,Y)-anonymity to include the data semantics as defined by a generalization hierarchy (e.g., VGH and DGH in Figure 4.1). In (X,Y,L)-anonymity, the values of attributes in X are associated to at least k values of Y at levels specified in L (w.r.t. VGHs of Y). For example, Table 4.3 is not (X,Y,L)-anonymous with $X = \{\mathsf{GEN}, \mathsf{AGE}, \mathsf{ZIP}\}$, $Y = \{\mathsf{MED}\}$, $k = 3$, and $L = \{l_1^{\mathsf{MED}}\}$. The values $(*, [31, 60], P*)$ in X are associated with a single value *NSAID* at level l_1^{MED} but it is (X,Y,L)-anonymous if $L = \{l_0^{\mathsf{MED}}\}$, e.g. $(*, [31, 60], P*)$ is linked to $k = 3$ values at level l_0^{MED}: *ibuprofen*, *naproxen* and *addaprin*.

Definition 4.6.2 ((X,Y,L)-anonymity)**.** *Consider a table R with schema $\mathcal{R}$ and attributes $X, Y \subseteq \mathcal{R}$, and a set of levels L corresponding to attribute* DGH*s from Y. R is* (X,Y,L)-anonymous *with value k if for every $t \in R$, there are at least k values in*

$G^t(R)$, *where* $G^t = \langle Q^t, L\rangle$ *is a GQ with* $Q^t(R) = \Pi_Y(\sigma_{X=t[X]}(R))$.

Intuitively, R is (X,Y,L)-anonymous if tuples in each X-group have at least k different values of Y at level L. (X,Y,L)-anonymity can apply tighter privacy restrictions compared to (X,Y)-anonymity by tuning the value of l (decided by the data owner) for a higher degree of privacy. Larger values of L make it more difficult to satisfy the (X,Y,L)-anonymity condition while (X,Y,L)-anonymity reduces to (X,Y)-anonymity if $l_i = 0$ for every $l_i \in L$. (X,Y,L)-anonymity is a semantic extension of (X,Y)-anonymity. Similar extensions can be defined for PPDP models such as (X,Y)-privacy, l-diversity, t-closeness, which all ignore the data semantics in the attribute DGHs and VGHs [66].

Pricing Generalized Queries. PPDP models have traditionally been used in non-interactive settings where a privacy-preserving table is published once. We take a user-centric approach and let users dictate the data they would like published. We apply PPDP in an interactive setting where values from relation R_{SP} are published incrementally according to (user) *CL* requests, while verifying that the disclosed data satisfies (X,Y,L)-anonymity. We adopt a data pricing scheme that assigns prices to GQs by extending the baseline data pricing algorithm defined in [70] to guarantee (X,Y,L)-anonymity.

Baseline Data Pricing. The baseline pricing model computes a price for a query Q over relation R according to the amount of information revealed about R when answering Q [70].

Given a query Q over a relation R (a database with single relation R), the baseline pricing model determines the price of Q based on the amount of information revealed about R by answering Q. Let $\mathcal{I}$ be a set of *possible relations* that the buyer believes to

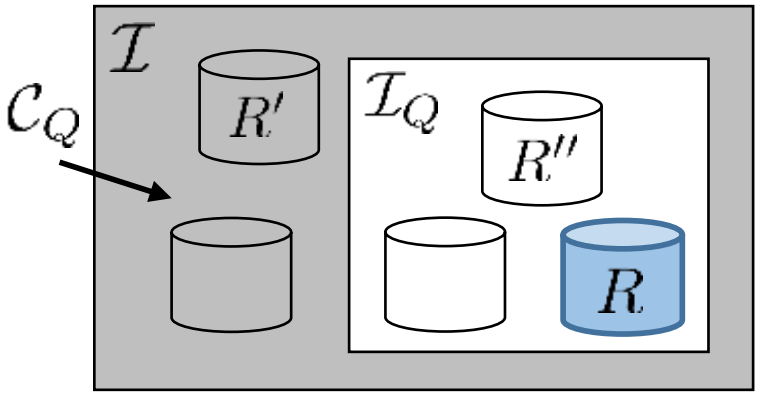


Figure 4.4: Possible relations $\mathcal{I}$, conflict set $\mathcal{C}_Q$ and admissible relations $\mathcal{I}_Q$ for query Q.

be R, representing his initial knowledge of R. As the buyer receives answers to $Q(R)$, he gains new knowledge, allowing him to eliminate relations R' from $\mathcal{I}$, which provide a different answer $Q(R') \neq Q(R)$. This set of eliminated instances R' is called the *conflict set of* Q denoted as $\mathcal{C}_Q$, and intuitively represents the amount of information that is revealed by answering Q (Figure 4.4). As more queries are answered, the size of $\mathcal{I}$ is reduced. We can apply a set function that uses $\mathcal{C}_Q$ to compute a price for Q. We can use the *weighted cover set* function with predefined weights assigned to the relations in $\mathcal{I}$. Query prices are computed by summing the weights for relations in $\mathcal{C}_Q$, which has been shown to give arbitrage-free prices [70]. In practice, the set $\mathcal{I}$ is usually infinite making it infeasible to implement. To circumvent this problem, a smaller, finite subset $\mathcal{S}$ called the *support set* is used to generate arbitrage-free prices [70]. The support set is defined as the neighbors of R, generated from R via tuple updates, insertions, and deletions. The values that are used to generate the support set are from the same domain of the original relation R.

Several optimizations are applied to the baseline pricing, and its effectiveness is experimentally justified [70]. Most importantly, the relations in the support set can

Algorithm 4: $price(Q, D, \mathcal{S}, w)$

Input : A query Q, a database D, support set $\mathcal{S}$, weight function w

Output: Price to answer Q

```
1 p ← 0;
2 for D' ∈ S do
3 |   if Q(D') ≠ Q(D) then p ← p + w(D') ;
4 end
5 return p;
```

be modeled by update operations. That is, we generate each relation in $\mathcal{S}$ from R by applying its corresponding update operation and use the resulting relation to compute the value of the weighted function as the final price. We roll-back the update to restore R, and continue this process to compute the weighted function w.r.t. other relations in $\mathcal{S}$. We avoid storing all databases in $\mathcal{S}$ to enable more efficient price computations.

Algorithm 4 provides pseudocode of the baseline algorithm. The algorithm takes query Q, database D, a support set $\mathcal{S}$, and a weight function w, and computes the price to answer Q over R. The baseline algorithm is history-aware as input $\mathcal{S}$ excludes databases that were already considered by past queries.

***SafePrice* Algorithm.** We propose *SafePrice* that enforces (X,Y,L)-anonymity over R (equivalently R_{SP} in our framework). We first present the definition of a *safe query*, i.e., criteria for a GQ to preserve (X,Y,L)-anonymity.

Definition 4.6.3. *(Safe Query) Consider a GQ G over a relation R with schema $\mathcal{R}$, $X, Y \subseteq \mathcal{R}$, and levels L corresponding to attributes in Y. Let $\mathcal{I}_G \subseteq \mathcal{I}$ be the set of relations R'' such that $G(R) = G(R'')$. G is safe (or preserves (X,Y,L)-anonymity*

of R) with value k, if for every tuple $t \in R$, there are at least k tuples in the set of answers $\{t'' \mid \exists R'' \in \mathcal{I}_G, t'' \in G^t(R'')\}$ *where* $G^t = \langle Q^t, L\rangle$ *is a GQ with* $Q^t(R'') = \Pi_Y(\sigma_{X=t[X]}(R''))$.

In Definition 4.6.3, $\mathcal{I}_G$ represents the set of relations that the buyer believes R is drawn from after observing the answer $G(R)$. If there are at least k tuples in the answer set of G^t over $\mathcal{I}_G$, this indicates that the buyer does not have enough information to associate the values in X to less than k values of Y at level L, thus preserving (X,Y,L)-anonymity. If the *SP* determines that a GQ is safe, he will assign a finite price relative to the amount of disclosed information about R.

Algorithm 5 presents the *SafePrice* details. The given support set $\mathcal{S}$ represents the user's knowledge about relation R after receiving the answer to past purchased queries. We maintain a set $\mathcal{S}_G$ that represents all admissible relations after answering G and captures the user's posterior knowledge about R after answering G. We use $\mathcal{S}_G$ to check whether answering G is safe. This is done by checking over instances in $\mathcal{S}_G$ whether values in X are associated with at least k values of Y using the query G^t. The price for a query is computed by summing the weights of the inadmissible relations in the conflict set $\mathcal{C}_G = \mathcal{S} \setminus \mathcal{S}_G$ (Line 4). Similar to the baseline pricing, we use the support set $\mathcal{S}$ and admissible relations $\mathcal{S}_G$ rather than $\mathcal{I}$ and $\mathcal{I}_G$, respectively. We iterate over tuples $t \in R$ (Line 5), and check whether values in X are associated with less than k values in Y over relations in $\mathcal{S}_G$. If so, we return an infinite price reflecting that query G is not safe (Line 8).

If the input GQ is not safe, Algorithm 5 preserves (X,Y,L)-anonymity by returning an infinite price.

Proof sketch: According to Definition 4.6.3, if G violates (X,Y,L)-anonymity then for

$t \in R$, there are less than k answers to G^t over relations in $\mathcal{I}_G$. Since $\mathcal{S}_G \subseteq \mathcal{I}_G$, there will be less than k answers to G^t over relations in $\mathcal{S}_G$, meaning Algorithm 5 assigns infinite price to G in Line 8. We note that if Algorithm 5 returns an infinite price, it does not imply G is unsafe (this only occurs when $\mathcal{S} = \mathcal{I}$). □

Algorithm 5: *SafePrice*$(G, R, \mathcal{S}, w)$

Input : G, R, $\mathcal{S}$, w

Output: Price of G

1 $p \leftarrow 0$; $\mathcal{S}_G \leftarrow 0$;
2 **for** $T \in \mathcal{S}$ **do**
3 **if** $G(T) = G(R)$ **then** $\mathcal{S}_G \leftarrow \mathcal{S}_G \cup \{T\}$;
4 **else** $p \leftarrow p + w(T)$;
5 **for** $t \in R$ **do**
6 $A \leftarrow \emptyset$;
7 **for** $R'' \in \mathcal{S}_G$ **do** $A \leftarrow A \cup G^t(R'')$;
8 **if** $|A| < k$ **then return** ∞;
9 **return** p;

Given the interactive setting between the *CL* and the *SP*, we must ensure that all (consecutively) disclosed values guarantee (X,Y,L)-anonymity over R. We ensure that *SafePrice* is history-aware by updating the support set $\mathcal{S}$ after answering each GQ. The *SP* uses the pricing function in Algorithm 5 to update $\mathcal{S}$ to reflect the current relations R' that have been eliminated by answering the latest GQ. The *SP* implements *AskPrice*(r_i, R_{SP}) (cf. Figure 4.3) by translating r_i to a GQ, and then invoking *SafePrice*.

In this work, we assume that requesting a query price is free. We acknowledge that

returning prices might leak information about the data being priced and purchased. This problem is discussed in the data pricing literature, particularly to incentivize data owners to return trustful prices when prices reveal information about the data (cf. [83] for a survey on this issue). We consider this problem as a a direction of future work.

4.6.3 Query Answering

A *CL* data request is executed via the $Pay(p, r_i, R_{SP})$ method, where she purchases the value in r_i at price p. The *Query Answering* module executes $Pay(p, r_i, R_{SP})$ via *SP* accepts payment, translates r_i to a GQ, and returns the answer over R_{SP} to *CL*. Lastly, *SP* updates the support set $\mathcal{S}$ to ensure *SafePrice* has an accurate history of disclosed data values.

We note that all communication between *CL* and *SP* is done via the *AskPrice* and *Pay* methods (provided by *SP*). Since our focus in this chapter is to develop a privacy-aware data cleaning framework, we assume there is a secure communication protocol between the *CL* and *SP*, and that all data transfer is protected and encrypted. Our current model is limited to a single *SP* that sells data at non-negotiable prices. We intend to explore general cases involving multiple *SP* providers and price negotiation as future work.

4.7 Data Cleaning with Generalized Values

Existing constraint-based data repair algorithms that propose updates to the data to satisfy a set of data dependencies assume an open-access data model with no data

privacy restrictions [60, 84, 85, 86, 87]. In these repair models, inadvertent data disclosure can occur as the space of repair candidates is not filtered nor transformed to obscure sensitive values. A privacy-aware data repair algorithm must address the challenge of providing an accurate repair to an error while respecting data generalizations and perturbations to conceal sensitive values.

We propose *SafeClean*, a data repair algorithm that resolves errors in a relation R_{CL} using data purchased from a service provider R_{SP}. The data disclosure in the service provider is limited by a fixed budget $\mathcal{B}$. The key distinctions of *SafeClean* from past work include: (i) *SafeClean* interacts with the service provider, *SP*, to purchase (possibly generalized), trusted values under a constrained budget $\mathcal{B}$. This eliminates the overhead of traversing a large search space of repair candidates; and (ii) *SafeClean* tries to obtain values with highest utility from *SP* for repairing *CL*. We present an overview of our cleaning algorithm and subsequently describe each component in detail.

4.7.1 Overview

For a fixed number of FDs Σ, the problem of finding minimal-cost data repairs to R_{CL} such that R_{CL} satisfies Σ is NP-complete [60]. Due to these intractability results, we necessarily take a greedy approach that cleans cell values in R_{CL} that maximally reduce the overall number of errors w.r.t. Σ. This is in similar spirit to existing techniques that have used various weighted cost functions [60, 17] or conflict hypergraphs [86] to model error interactions among data dependencies in Σ.

Given R_{CL} and Σ, we identify a set of error cells that belong to tuples that falsify some $\sigma \in \Sigma$. For an error cell $e \in \mathcal{E}$, we define eq(e) as the equivalence class to which

e belongs. An equivalence class is a set of database cells with the same value such that Σ is satisfied [60]. We repair w.r.t. eqs for two reasons: (i) by clustering cells into eqs, we determine a repair value for a *group of cells* rather than an individual cell, thereby improving performance; and (ii) we utilize every cell value within an eq to find the best repair.

The *SafeClean* algorithm repairs FD errors by first finding all the eqs in R_{CL}. The algorithm then iteratively selects an eq, and purchases the true value of a cell, and updates all dirty cells in the same class to the purchased value. At each iteration, we repair the eq class with cells that participate in the largest number of FD errors. At each iteration, *SafeClean* assigns a portion of the budget $\mathcal{B}$ that is proportional to the number of errors relative to the total number of errors in R_{CL}. *SafeClean* continues until all the eqs are repaired, or the budget is exhausted.

SafeClean Algorithm. Algorithm 6 gives details of SafeClean's overall execution. The algorithm first generates the set of eqs via *GenerateEQs* in Line 2. Equivalence classes containing only one value are removed since there is no need for repair. In Line 7, *SafeClean* selects an equivalence class eq_i with cells participating in the largest number of violations for repair (further details in Section 4.7.3). To repair the error cells in eq_i, *SafeClean* generates a request r_i using *GenerateRequest* that requests a repair value for a cell in eq_i (Line 8). This request is made at the lowest possible level (less than l_{max}) at a price allowable within the given budget. The algorithm assigns a fraction of the remaining budget, i.e. $\alpha_i \times B$ ($B \leq \mathcal{B}$ is the remaining budget) to purchase data at each iteration. This fraction depends on the number of violations in eq_i (cf. Section 4.7.4 for details). If such a request can be satisfied, the value(s) are purchased and applied (Lines 10-13). If there is an insufficient budget remaining

Algorithm 6: *SafeClean*$(R_{CL}, R_{SP}, \Sigma, \mathcal{B})$

Input : R_{CL}, R_{SP}, Σ, $\mathcal{B}$

Output: Clean R'_{CL}

$R'_{CL} \leftarrow R_{CL}$;

$EQ \leftarrow GenerateEQs(R'_{CL}, \Sigma)$;

$B \leftarrow \mathcal{B}$;

for $eq \in EQ$ **do**

  if $Resolved(eq)$ **then** $EQ \leftarrow EQ \setminus \{eq\}$;

while $B > 0$ **and** $EQ \neq \emptyset$ **do**

  $eq \leftarrow Select(EQ)$;

  $r_i \leftarrow GenerateRequest(eq, \alpha_i \times B, R_{SP})$;

  if $r_i \neq null$ **then**

    $p_i \leftarrow AskPrice(r_i, R_{SP})$;

    $u_i \leftarrow Pay(p_i, r_i, R_{SP})$;

    $B \leftarrow B - p_i$;

    $ApplyRepair(eq, u_i)$;

  $EQ \leftarrow EQ \setminus eq$;

return R'_{CL}

to purchase a repair value, then the eq cannot be repaired. In either case, *SafeClean* removes eq_i from EQ and continues with the next eq (Line 14). *SafeClean* terminates when $\mathcal{B}$ is exhausted, or there are no eqs are remaining. We present details of eq generation, selection, request generation, and data purchase/repair in the following sections.

4.7.2 Generating Equivalence Classes

The equivalence classes are generated by *GenerateEQs* in Algorithm 7 that takes as input R_{CL} and Σ, and returns the set of equivalence classes EQ. For every cell $c_i \in R_{CL}$, the procedure initializes the equivalence classes of c_i as $eq(c_i) = \{c_i\}$, and adds it to the set of equivalence classes EQ (Line 2). We then iteratively merge the equivalence classes of any pair of cells $c_1 = t_1[B], c_2 = t_2[B]$ if there is a FD $\varphi : A \rightarrow B \in \Sigma$, $t_1[A] = t_2[A]$, and both $t_1[A]$ and $t_2[A]$ are ground. The procedure stops and returns EQ when no further pair of equivalence classes can be merged.

Algorithm 7: *GenerateEQs*(R_{CL}, Σ)

Input : R_{CL}, Σ,

Output: The set of equivalence classes EQ

1 $EQ \leftarrow \emptyset$;

2 **for** $c_i \in R_{CL}$ **do** $EQ \leftarrow EQ \cup \{\{c_i\}\}$;

3 **for every** $t_1, t_2 \in R_{CL}$ **do**

4 $\quad eq \leftarrow MergeEQ(t_1, t_2, \Sigma)$

5 $\quad EQ \leftarrow EQ \cup eq$

6 **end**

7 **return** *EQ;*

Algorithm 8: *MergeEQ*

Input : Tuples t_1, t_2 and Σ

Output: A equivalence class eq

```
1 eq ← ∅;
2 for and φ : A → B ∈ Σ do
3 |   if t1[A] = t2[A] then eq ← eq ∪ (t1, t2) ;
4 end
5 return eq;
```

Example 4.7.1. Given FD $\varphi : [\mathsf{GEN}, \mathsf{DIAG}] \rightarrow [\mathsf{MED}]$ in Table 4.2, since t_1, t_2 and t_3 have the same attribute values on GEN and DIAG, we merge $t_1[\mathsf{MED}]$, $t_2[\mathsf{MED}]$ and $t_3[\mathsf{MED}]$ into the same EQ_1. Similarly, we cluster t_4 and t_5 into the same EQ_2.

4.7.3 Selecting Equivalence Classes for Repair

In each iteration of Algorithm 6, we repair the cells in an eq eq_i that will resolve the most errors in R_{CL} w.r.t. Σ. To achieve this goal, we choose eq_i as the eq with cells participating in the largest number of errors. For a cell $c_j \in R_{CL}$ and an FD $\varphi : A \rightarrow B \in \Sigma$, let $\mathcal{E}(R_{CL}, \varphi, c_j)$ be the set of errors $\{t_1, t_2\}$ w.r.t. φ and $c_j \in \{t_1[A], t_1[B], t_2[A], t_2[B]\}$. For an eq eq_i, let $\mathcal{E}(R_{CL}, \varphi, eq_i) = \bigcup_{c_j \in eq_i} \mathcal{E}(R_{CL}, \varphi, c_j)$ and $\mathcal{E}(R_{CL}, \Sigma, eq_i) = \bigcup_{\varphi \in \Sigma} \mathcal{E}(R_{CL}, \varphi, eq_i)$. The eq eq_i returned by *Select* in Line 2 of Algorithm 6 is the eq with the largest number of errors in $\mathcal{E}(R_{CL}, \Sigma, eq_i)$. In other words, this is the number of tuple pairs that every cell in eq_i participates, summed over all FDs in Σ.

Example 4.7.2. Continue with EQ_1 and EQ_2 in Example 4.7.1. According to our

error definition, given FD $\varphi : [\mathsf{GEN}, \mathsf{DIAG}] \rightarrow [\mathsf{MED}]$, EQ_1 gets involved in three errors ($\{t_1, t_2\}$, $\{t_1, t_3\}$ and $\{t_2, t_3\}$), while EQ_2 has one error ($\{t_3, t_4\}$), so our algorithm will select EQ_1 to repair first.

4.7.4 Data Request Generation

To repair cells in eq_i, we generate a request $r_i = \langle c, l \rangle$ by *GenerateRequest* (Algorithm 9) that requests the accurate and trusted value of a cell $c \in eq_i$ at level l from R_{SP}. There are two restrictions on this request: (i) its price has to be within the budget $\alpha_i \times \mathcal{B}$, and (ii) the level l has to be $\leq l_{max}$. The value α_i is defined as follows:

$$\alpha_i = \frac{\mathcal{E}(R_{CL}, \Sigma, eq_i)}{\sum_{eq_j \in EQ} \mathcal{E}(R_{CL}, \Sigma, eq_j)}.$$

In this definition, the allocated budget $\alpha_i \times B$ to each iteration is proportional to the number of FD violations in eq_i, and also depends on the total number of errors in R_{CL}. This allocation model improves upon previous work that decrease the budget allocated to the i^{th}-request by a factor of $\frac{1}{i}$, and does not adjust the allocation to the number of errors in which a cell participates [11]. Note that if the price paid for r_i (i.e. p_i) is less than this allocated budget, the remaining budget carries to the next iteration through B.

If there is no such request, *GenerateRequest* returns null, indicating that eq_i cannot be repaired with the allocated budget (Line 10). If there are several requests that satisfy (i) and (ii), we follow a greedy approach and select the request at the lowest level with the price to break ties (Line 5). For a cell $c_i \in eq_i$, *LowestAffordableLevel* in Line 3 finds the lowest level in which the value of c_i can be purchased from R_{SP}

considering the restrictions in (i) and (ii). Our greedy algorithm spends most of the allocated budget $\alpha_i \times B$ in the current iteration. An alternative approach can select requests with the highest acceptable level (l_{max}) to preserve its budget for future iterations.

Algorithm 9: *GenerateRequest*(C, b, R_{SP})

Input : eq C, budget b and R_{SP}.

Output: Request r_i

```
1  l ← l_h; p ← ∞; c ← null;
2  for c_i ∈ C do
3  |   l_i ← LowestAffordableLevel(c_i, b, l_max, R_SP);
4  |   p_i ← AskPrice(⟨c_i, l_i⟩, R_SP);
5  |   if l_i < l or (l_i == l and p_i ≤ p) then
6  |   |   c ← c_i; l ← l_i; p ← p_i;
7  |   end
8  end
9  if c ≠ null then return ⟨c, l⟩;
10 return null;
```

4.7.5 Purchase Data and Repair

To repair the cells in eq_i, Algorithm 6 invokes $Pay(r_i, R_{SP})$ to purchase the trusted value u_i and replaces the value of every cell in eq_i with u_i in *ApplyRepair*. The algorithm then removes the eq eq_i from EQ and continues to repair the next eq. It stops when there is no eq to repair or the budget is exhausted.

Example 4.7.3. Continuing from Example 4.7.2, since EQ_1 has the largest number of errors, we purchase trusted value from R_{SP} to repair EQ_1 first. If the purchased value is general value *NSAID*, we update all cells in EQ_1, which are $t_1[\mathsf{MED}]$, $t_2[\mathsf{MED}]$ and $t_3[\mathsf{MED}]$ to *NSAID* to resolve the inconsistency.

4.7.6 Complexity Analysis

We analyze the complexity of *SafeClean*'s modules:

- Error identification is based on selecting the dirtiest cell e, and resolved w.r.t. an equivalence class. In the worst case, this is quadratic in the number of tuples in R_{CL}.
- For each data request to fix an error, executing *SafePrice* and *pay* rely on R_{SP}, the support set $\mathcal{S}$, and the complexity of GQ answering. We assume the size of $\mathcal{S}$ is linear w.r.t the size of R_{SP}. The complexity of running GQs is the same as running SQL queries. Thus, all procedures run in polynomial time w.r.t. the size of R_{SP}.
- In the *applyRepair* grounding process, we must ground each returned value for each error w.r.t. each FD, taking time on the order of $\mathcal{O}(|\mathcal{E}||A||\Sigma|)$ for attribute domain size $|A|$. Updating the affected cells in the equivalence class, and their dirty scores are both bounded by the number of cells in R_{CL}. Hence, Algorithm 6 runs in polynomial time w.r.t. the size of R_{SP} and R_{CL}.

4.8 Experiments

We now turn to the evaluation of *SafePrice* and *SafeClean* using three real datasets. Our evaluation focuses on the following objectives:

1. The efficiency and effectiveness of *SafePrice* to generate reasonable prices that allow *SafeClean* to effectively repair the data.

2. We evaluate the impact of generalized values on the repair error, and the runtime performance. In addition, we study the proportion of generalized repair values that are recommended for varying budget levels.

3. We evaluate the repair error and scalability of *SafeClean* as we vary the parameters k, l, e and $\mathcal{B}$ to study the repair error to runtime tradeoff.

4. We compare *SafeClean* against PrivateClean, a framework that explores the link between differential privacy and data cleaning [64]. We study the repair error to runtime tradeoff between these two techniques, and show that data randomization in PrivateClean significantly hinders error detection and data repair, leading to increased repair errors.

4.8.1 Experimental Setup

We implement *PACAS* using Python 3.6 on a server with 32 Core Intel Xeon 2.2 GHz processor with 64GB RAM. We describe the datasets, baseline comparative algorithm, metrics and parameters.

Parameters. We vary the following parameters to evaluate the performance and scalability of our algorithms: (i) budget $\mathcal{B}$; (ii) the size of the support set $|\mathcal{S}|$ in the pricing function; (iii) l the level of generalization; (v) k that controls the number of tuples in each X-group; and (vi) the error rate e in R_{CL}. The default parameters are showed in Table 4.6.

Datasets We use three real datasets. We selected these datasets because they are

	Clinical	**Census**	**Food**
$\|R_{CL}\|$	345,000	300,000	30,000
n	29	40	11
$\|\Pi_A(R)\|$	61/73	17/250	248/70
$\|\mathsf{VGH}^A\|$	5/5	5/6	5/5

Table 4.5: Data characteristics.

Sym.	**Description**	**Values**
$\mathcal{B}$	budget	0.2, 0.4, 0.6, **0.8**
$\|\mathcal{S}\|$	support set size	6, 8, **10**, 12, 14
l	generalization level	0, 1, **2**, 3, 4
k	#tuples in X-group	1, 2, **3**, 4, 5
e	error rate	0.05, **0.1**, 0.15, 0.2, 0.25

Table 4.6: Parameter values (defaults in bold).

widely used in the existing data cleaning work such as [15, 16, 17, 20, 21, 64]. Table 4.5 gives the data characteristics, showing a range of data sizes in the number of tuples ($|R_{CL}|$), number of attributes (n), number of unique values in the sensitive attribute A ($|\Pi_A(R)|$), and the height of the attribute VGH^A ($|\mathsf{VGH}^A|$). We denote sensitive attributes with an asterisk (*).

Clinical Trials (Clinical). The Linked Clinical Trials database describes patient demographics, diagnosis, prescribed drugs, symptoms, and treatment [88]. We select the country, gender, source and age as QI attributes. We use the same FDs as the the existing work [17]. These two FDs are: (i) φ_1 : [age, overall_status, gender] $\rightarrow$

[drug_name*]; and (ii) φ_2 : [overall_status, timeframe, measure] $\rightarrow$ [condition*]. We construct attribute value generalization hierarchies (VGH) with five levels on attributes drug name and condition, respectively using external ontologies (Bioprotal Medical Ontology [89], the University of Maryland Disease Ontology [90], and the Libraries of Ontologies from the University of Michigan [91]). The average number of children per node in the VGH is eight.

Census. The U.S. Census Bureau provides population characteristics such as education level, years of schooling, occupation, income, and age [92]. We select sex, age, race, and native country as QI attributes. We define two FDs: (i) φ_3 : [age, education-num] $\rightarrow$ [education*]; and (ii) φ_4 : [age, industry code, occupation] $\rightarrow$ [wage-per-hour*]. We uses these two FDs because normally age and education number can determine the education level of a person and age, industry, occupation can determine the wage of a individual per hour. We construct VGH on attributes wage-per-hour and education by stratifying wage and education levels according to hierarchies from US statistics [93], and the US Department of Education [94]. The average number of children per node is five.

Food Inspection (Food). This dataset contains violation citations of inspected restaurants in New York City describing the address, borough, zipcode, violation code, violation description, inspection type, score, grade. We define inspection type, borough, grade as QI attributes. We define two FDs: (i) σ_5 : [borough, zipcode] $\rightarrow$ [address*]; and (ii) σ_6 : [violation code, inspection type] $\rightarrow$ [violation description*]. We use these two FDs because borough and zipcode can identify an unique address; violation code and inspection type can determine the violation description uniquely as well. We construct attribute VGH on address and violation description by classifying streets into

neighborhoods, districts, etc, and extracting topic keywords from the description and classifying the violation according to the Food Service Establishment Inspection Code [95]. The average number of children per node in the VGH is four.

For each dataset, we manually curate a clean instance R_{SP} according to the defined FDs, verify with external sources and ontologies, and is used as the ground truth. To create a dirty instance R_{CL}, we duplicate R_{SP} to obtain R_{CL}, and use BART, an error generation benchmarking tool for data cleaning applications to inject controlled errors in the FD attributes [96]. We use BART to generate constraint-induced errors and random errors, and define error percentages ranging from 5% to 25%, with respect to the number of tuples in a table. We use BART's default settings for all other parameters.

Comparative Baseline. The closest comparative baseline is PrivateClean, a framework that explores the link between differential privacy and data cleaning [64]. PrivateClean provides a mechanism to generate ϵ-differentially private datasets on numerical and discrete values from which a data analyst can apply data cleaning operations. PrivateClean proposes randomization techniques for discrete and numeric values, called Generalized Randomized Response (GRR), and applies this across all attributes. To guarantee a privatized dataset with discrete attributes is ϵ-differentially private at confidence $(1-\alpha)$, a sufficient number of distinct records is needed. In our comparison evaluation, we set $\alpha = 0.05$, and the degree of privacy $p = 0.5$ (applied uniformly across the attributes and equivalent to ϵ for discrete attributes [64]). Since PrivateClean does not directly propose a data cleaning algorithm, but rather data cleaning operators, we apply the well-known Greedy-Repair [60] FD repair algorithm to generate a series of updates to correct the FD violations. We use the transform()

Method	δ
PrivateClean [64]	342k
SafeClean [12]	183k

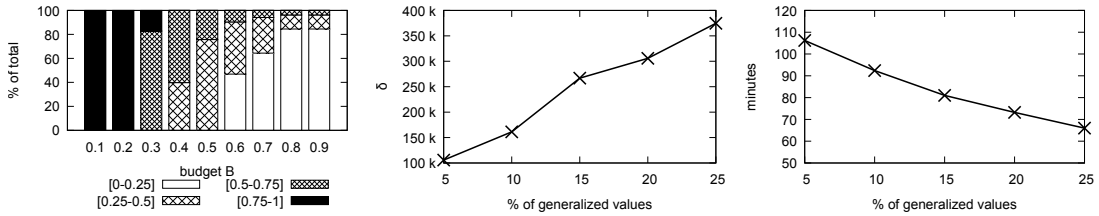
Table 4.7: Comparison Summary

operation to represent each of these updates in PrivateClean, and use the source code provided by the authors in our implementation. We choose the Greedy-Repair algorithm for its similarity to our repair approach; namely, to repair cells on an equivalence class basis, and to minimize a cost function that considers the number of data updates and the distance between the source and target values.

Metrics. We compute the average runtime over four executions. To measure the quality of the recommended repairs, we define the *repair error* of a cell as the distance between a cell's true value and its repair value. We use the semantic distance measure, $\delta(v, v')$, in Section 4.4.1, that quantifies the distance between two values v (suppose a true value), and v' (a repair value) in the attribute value generalization hierarchy, and considers the distribution of v and v' in the relation. We assume a cell's value, and its repair are both in the generalization hierarchy. The repair error of a relational instance is computed as the sum of the repair errors across all cells in the relation. We use the absolute value of the repair error in our experiments (denoted as δ). Similar to other error metrics (such as mean squared error), lower error values are preferred.

4.8.2 Generalized Values

We measure the proportion of generalized values that are returned for varying levels of the budget $\mathcal{B}$. Since a repair value may be returned at any level of the VGH, we



(a) Ratio of gen. values. (b) Error vs. gen. values. (c) Runtime vs. gen. values.

Figure 4.5: Evaluation on gen. values

compute the semantic distance between the generalized value v and the corresponding ground value v' in the *CL*. We normalize the distance between (v, v') into four ranges: [0-0.25], [0.25-0.5], [0.5-0.75], [0.75-1], where a distance of zero indicates v and v' are both ground values.

Figure 4.5a shows the relative proportions for varying $\mathcal{B}$ values over the clinical dataset. As expected, the results show that the proportion of generalized values at the highest levels of the VGH occur for low $\mathcal{B}$ values since we can only afford (cheaper) generalized values under a constrained budget. In contrast, for $\mathcal{B}$ values close to 0.9, close to 85% of the repair values are specific, ground values with a distance range of [0, 0.25], while the remaining 15% are generalized values at the next level. We observe that for $\mathcal{B} > 0.6$ approximately 70% of the total repair values are very close to the ground value (where distance is at [0-0.25]), indicating higher quality repairs.

We evaluate the impact on the runtime to repair error tradeoff when an increasing number of generalized values occur in the repaired relation. We control the number of generalized values indirectly via the number of error cells under a constrained budget $\mathcal{B} = 0.1$ where it is expected that close to all repair recommendations will be general values. Figure 4.5b and Figure 4.5c show the repair error and runtime curves over the clinical data, respectively. As expected, we observe for an increasing

number of generalized values, the repair error increases as the constrained budget leads to more values returned at the highest * generalized value, thereby increasing the distance between ground-truth relation and repaired relation. In contrast, the increased number of generalized values leads to lower runtimes due to the increased number of unsatisfied query requests.

4.8.3 *SafePrice* Efficiency and Effectiveness

The *SafePrice* algorithm relies on a support set $\mathcal{S}$ to determine query prices by summing the weights of discarded instances from the conflict set $\mathcal{C}$. These discarded instances represent the knowledge gained by *CL*. We vary the size of the initial $\mathcal{S}$ to determine its influence on the repair error (δ), and the overall runtime. Figure 4.6 shows a steady decrease in the repair error for increasing $|\mathcal{S}|$. As $|\mathcal{S}|$ grows, the *SP* is less restrictive to answer GQs and fewer requests are declined at lower levels. As more requests are answered at these lower levels, the repair error decreases. Figure 4.7a shows that the *SafePrice* runtime scales linearly with increasing $|\mathcal{S}|$, making it feasible to implement in practice. From Figures 4.6 and 4.7a, we determine that *SafePrice* achieves an average 6% reduction in the repair error at a cost of 16m runtime. This is expected due to the additional time needed to evaluate the larger space of instances to answer GQs. Comparing across the three datasets, the data sizes affect runtimes as a larger number of records must be evaluated during query pricing. This is reflected in longer runtimes for the larger clinical and census datasets.

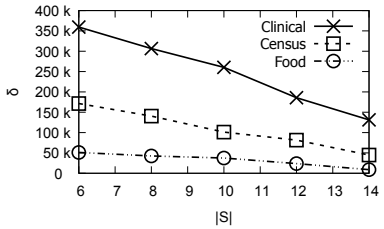
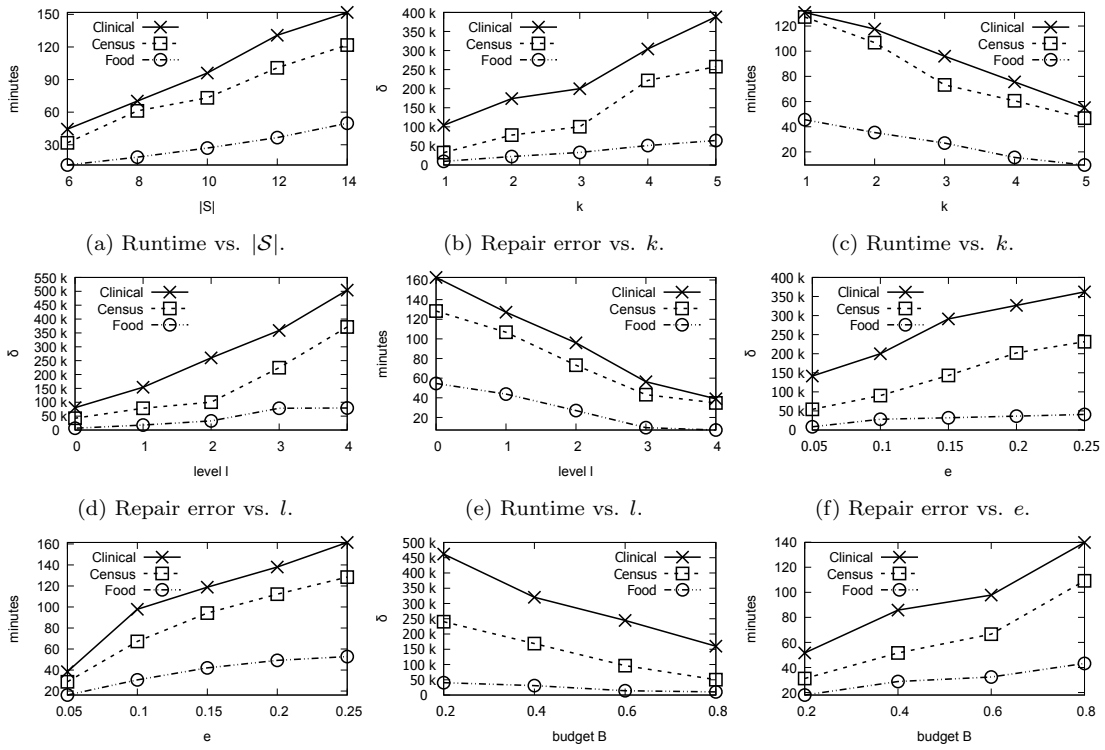


Figure 4.6: Repair error vs. $|\mathcal{S}|$.



(a) Runtime vs. $|\mathcal{S}|$. (b) Repair error vs. k. (c) Runtime vs. k.

(d) Repair error vs. l. (e) Runtime vs. l. (f) Repair error vs. e.

(g) Runtime vs. e. (h) Repair error vs. $\mathcal{B}$. (i) Runtime vs. $\mathcal{B}$.

Figure 4.7: *SafePrice* Parameter Sensitivity

4.8.4 *SafePrice* Parameter Sensitivity

We vary the parameters k, GQ level l, error rate e, budget $\mathcal{B}$, and measure their influence on *SafeClean* repair error and runtime over all three datasets. We expect that enforcing more stringent privacy requirements through larger k and l values will result in larger repair errors. Figures 4.7b to 4.7e do indeed reflect this intuition.

In Figure 4.7b, *SafeClean* experiences larger repair errors for increasing k as generalizations to conceal sensitive values become increasingly dependent on the attribute domain and its VGH i.e., (X,Y,L)-anonymity indicates there must be at least k values at a given level l in the VGH. Otherwise, the data request is denied. Figure 4.7c shows that for increasing k, runtimes decrease linearly as query requests to satisfy

more stringent k become more difficult. On average, we observe that an approximate 10% improvement in runtime leads to a 7% increase in the repair error for each increment of k. Figures 4.7d and 4.7e show the repair error and runtime, respectively, as we vary the query level parameter l. The repair error increases, particularly after $l = 3$ as more stringent privacy requirements are enforced, i.e., l distinct values are required at each generalization level. This makes satisfying query requests more difficult, leading to unrepaired values and lower runtimes, as shown in Figure 4.7e.

Figure 4.7f shows the repair error δ increases as we scale the error rate e. For a fixed budget, increasing the number of FD errors leads to a decreasing budget for each FD error. This makes some repairs unaffordable for the *CL*, leading to unrepaired values and an increased number of generalized repair values. This situation can be mitigated if we increase the budget $\mathcal{B}$. As expected, Figure 4.7g shows that the *SafeClean* runtime increases for an increasing number of FD violations, due to the larger overhead to compute more equivalance classes, compute prices to answer queries, and to check consistency in the *CL*.

Figures 4.7h and 4.7i show the repair error and runtime, respectively, as we vary the budget $\mathcal{B}$. For increasing budget allocations, we expect the *SP* to recommend more ground (repair) values, and lower repair error values, as shown in Figure 4.7h. Given the larger budget allocations, the *SP* is able to answer a larger number of query requests, and must compute their query prices, thereby increasing algorithm runtime. We observe that an average 14% reduction in the repair error leads to an approximate 7% increase in runtime.

We compare *SafeClean* to PrivateClean, a framework for data cleaning on locally differentially private relations [64]. Section 4.8.1 describes the baseline algorithm

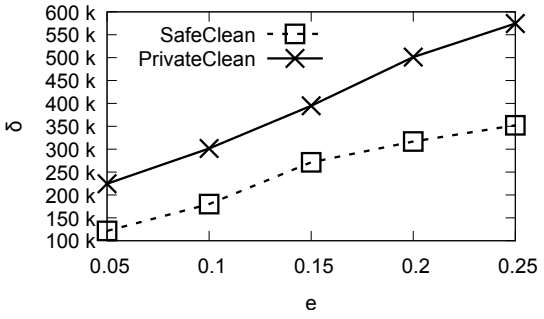


Figure 4.8: Comparative repair error.

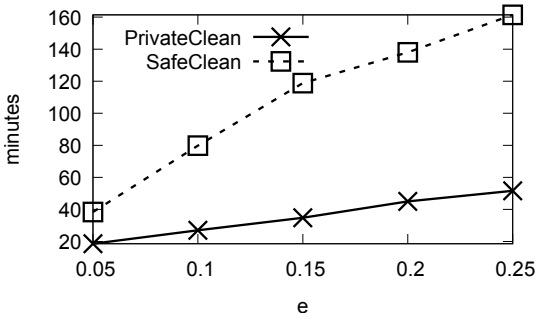


Figure 4.9: Comparative runtime.

parameter settings and configuration. Despite the differing privacy models, our evaluation aims to explore the influence of increasing error rates on the repair error δ, and algorithm runtimes using the Clinical dataset. We hope these results are useful for practitioners to understand qualitative and performance trade-offs between the two privacy models. For *SafeClean*, we measure total time of the end-to-end process from error detection to applying as many repairs as the budget allows. In PrivateClean, we measure the time to privatize the R_{CL}, error detection, running the Greedy-Repair FD repair algorithm [60], and applying the updates via the Transform operation.

For PrivateClean, we measure the repair error $\delta(v, v')$ for source value v and target (clean) value v', as recommended by Greedy-Repair, where both v, v' are ground values.

Figure 4.8 shows the comparative repair error between *SafeClean* and PrivateClean as we vary the error rate e. *SafeClean* achieves an average -41% lower repair error than PrivateClean. This poor performance by PrivateClean is explained by the underlying data randomization used in differential privacy, which provides strong privacy guarantees, but poor data utility, especially in data cleaning applications. As acknowledged by the authors, identifying and reasoning about errors over randomized response data is hard [64]. This randomization may update rare values to be

more frequent, and similarly, common values to be more rare. This leads to more uniform distributions (where stronger privacy guarantees can be provided), but negatively impact error detection and cardinality-minimal data repair techniques that rely on attribute value frequencies to determine repair values [60]. We envision that *SafeClean* is a compromise towards providing an (X,Y,L)-anonymous instance while preserving data utility.

SafeClean's lower repair error comes at a runtime cost, as shown in Figure 4.9. As we scale the error rate, *SafeClean*'s runtime scales linearly due to the increased overhead of data pricing. Recall the pricing mechanism in Section 4.6.2, the price of query Q is determined by the query answer over the relation D and its neighbors D' in the support set $\mathcal{S}$. In contrast, PrivateClean does not incur such an overhead due to its inherent data randomization. There are opportunities to explore optimizations to lower *SafeClean*'s overhead to answer query requests and compute prices to answer each query. In practice, optimizations can be applied to improve overall performance, including decreasing the parameter k according to application requirements, and considering distributed (parallel) executions of query processing over partitions of the data. *SafeClean* aims to provide a balanced approach towards achieving data utility for data cleaning applications while protecting sensitive values via data pricing and (X,Y,L)-anonymity.

4.9 Conclusions

We present PACAS, a data cleaning-as-a-service framework that preserves (X,Y,L)-anonymity in a service provider with sensitive data, while resolving errors in a client data instance.

PACAS anonymizes sensitive data values by implementing a data pricing scheme that assigns prices to requested data values while satisfying a given budget. We propose generalized repair values as a mechanism to obfuscate sensitive data values, and present a new definition of consistency with respect to functional dependencies over a relation. To adapt to the changing number of errors in the database during data cleaning, we propose a new budget allocation scheme that adjusts to the current number of unresolved errors. We believe that PACAS provides a new approach to privacy-aware cleaning that protects sensitive data while offering high data cleaning utility, as data markets become increasingly popular. As next steps, we are investigating: (i) optimizations to improve the performance of the data pricing modules; and (ii) extensions to include price negotiation among multiple service providers and clients.

In this chapter, we introduce a privacy-preserving data cleaning framework that considers data privacy together with data cleaning through the anonymization approach. However, the anonymization approach fails to provide any data diversity guarantee, which is desirable in many applications. In the next chapter, we study the problem of finding a diverse anonymized data instance which ensures data diversity during the anonymization process.

Chapter 5

Diversifying Anonymized Data with Diversity Constraints

5.1 Introduction

Organizations often share user information with third parties to analyze collective user behaviour and for targeted marketing. For example, in the pharmaceutical industry, hospital and medical records are shared and sold to data brokers who aggregate longitudinal data from patient records, insurance claims and lab tests to derive collective insights for research and drug development. Protecting user privacy is critical to safeguard personal and sensitive data. The European Union General Data Protection Regulation (GDPR), and variants such as the California Consumer Protection Act (CCPA) aim to control how organizations manage user data. For example, a major tenet in GDPR is *data minimization* that states companies should collect and share only a minimal amount of personal data sufficient for their purpose. CCPA takes this one step further requiring companies to document and track onward transfer of data

to third parties. Given the impossibility of knowing how a published data instance will be used in the future, determining a minimal amount of personal data to share is a challenge.

One solution is to apply differential privacy techniques to the entire data instance that provide provable guarantees. These guarantees often rely on aggregation queries over sufficiently large samples such that the output is not influenced by the presence (or absence) of any single record [69]. Unfortunately, applications often experience poor data utility and accuracy due to the necessary data randomization in differential privacy. Privacy-preserving data publishing (PPDP) provides a middle-ground to safeguard individual privacy while ensuring the published data remains practically useful for subsequent analysis. One of the benefits of PPDP is the focus on publishing actual data, rather than statistical summaries and relationships about the data. Anonymization is the most common form of PPDP, where quasi-identifiers and/or sensitive values are obfuscated via suppression or generalization [66].

As anonymized instances are shared with third parties for decision making and analysis, there is growing interest to ensure that data (and the algorithms that generate and use the data) are diverse and fair. Diversity is a rather established notion in data analytics that refers to the property of a selected set of individuals. Diversity requires the selected set to have a minimum representation from each group of individuals [97, 7] while determining the minimum bound for each group is often domain and user dependent.

To avoid biased decision making, incorporating diversity into computational models is essential to prevent and minimize discrimination against disadvantaged and minority groups. In this chapter, we focus on diversity, and study how diversity

ID	GEN	ETH	AGE	PRV	CTY	DIAG
t_1	Female	Caucasian	80	AB	Calgary	Hypertension
t_2	Female	Caucasian	32	AB	Calgary	Tuberculosis
t_3	Male	Caucasian	59	AB	Calgary	Osteoarthritis
t_4	Male	Caucasian	46	MB	Winnipeg	Migraine
t_5	Male	African	31	MB	Winnipeg	Hypertension
t_6	Male	African	43	BC	Vancouver	Seizure
t_7	Male	Caucasian	29	BC	Vancouver	Hypertension
t_8	Female	Asian	58	BC	Vancouver	Seizure
t_9	Female	Asian	47	MB	Winnipeg	Influenza
t_{10}	Female	Asian	71	BC	Vancouver	Migraine

Table 5.1: Medical records relation (R)

requirements can be modeled and satisfied in PPDP. In PPDP, non-diverse data instances that exclude minority group give an inaccurate representation of the population in subsequent data analysis. Unfortunately, early PPDP work [98, 99, 66, 6, 26], and recent work on PPDP for linked data and graphs [100, 101] have not studied techniques to include diversity in published data instances. Consider the following example demonstrating the challenges of applying diversity in PPDP.

Example 5.1.1. Table 5.1 shows relation R containing patients medical records describing gender (GEN), ethnicity (ETH), age (AGE), province (PRV), city (CTY), and diagnosed disease (DIAG). Third-parties such as pharmaceuticals, insurance firms are interested in an anonymized R containing patients from diverse geographies, gender, and ethnicities. Let GEN, ETH, AGE, CTY, PRV, be quasi-identifier (QI) attributes,

ID	GEN	ETH	AGE	PRV	CTY	DIAG
r_1	$\star$	Caucasian	$\star$	AB	Calgary	Hypertension
r_2	$\star$	Caucasian	$\star$	AB	Calgary	Tuberculosis
r_3	$\star$	Caucasian	$\star$	AB	Calgary	Osteoarthritis
r_4	Male	$\star$	$\star$	$\star$	$\star$	Migraine
r_5	Male	$\star$	$\star$	$\star$	$\star$	Hypertension
r_6	Male	$\star$	$\star$	$\star$	$\star$	Seizure
r_7	Male	$\star$	$\star$	$\star$	$\star$	Hypertension
r_8	Female	Asian	$\star$	$\star$	$\star$	Seizure
r_9	Female	Asian	$\star$	$\star$	$\star$	Influenza
r_{10}	Female	Asian	$\star$	$\star$	$\star$	Migraine

Table 5.2: Anonymized relation with $k = 3$

and let DIAG be a sensitive attribute. Existing PPDP methods such as k-anonymity prevent re-identification of an individual along the QI attributes from $k - 1$ other tuples. Table 5.2 shows a k-anonymized instance for $k = 3$ where tuples are clustered along the QI attributes via value suppression [6, 26].

The k-anonymization problem is to generate a k-anonymous relation through an anonymization process, such as generalization and suppression, while incurring minimum information loss. Suppression replaces some QI attribute values with $\star$s to achieve k-anonymity. There are several measures of information loss in PPDP [66], e.g., counting the number of $\star$s. Existing k-anonymization techniques do not preserve diversity in R since these information loss measures do not capture diversity semantics. □

ID	GEN	ETH	AGE	PRV	CTY	DIAG
s_1	Female	Caucasian	$\star$	AB	Calgary	Hypertension
s_2	Female	Caucasian	$\star$	AB	Calgary	Tuberculosis
s_3	Male	Caucasian	$\star$	$\star$	$\star$	Osteoarthritis
s_4	Male	Caucasian	$\star$	$\star$	$\star$	Migraine
s_5	Male	African	$\star$	$\star$	$\star$	Hypertension
s_6	Male	African	$\star$	$\star$	$\star$	Seizure
s_7	$\star$	$\star$	$\star$	BC	Vancouver	Hypertension
s_8	$\star$	$\star$	$\star$	BC	Vancouver	Seizure
s_9	Female	Asian	$\star$	$\star$	$\star$	Influenza
s_{10}	Female	Asian	$\star$	$\star$	$\star$	Migraine

Table 5.3: Anonymized relation with $k = 2$.

Unfortunately, existing methods fail to provide any diversity guarantees in published, privatized data instances. This leads to inaccurate and biased decision making in downstream data analysis. For example, in health care, anonymized patient records that exclude minority groups or fail to preserve the ratios of patients across different diseases misrepresent the true patient population, causing insufficient resource allocations.

To model diversity, existing work have proposed declarative methods in the form of *diversity constraints* [7], which define the expected frequencies that sensitive values in the data must satisfy. Using k-anonymity as our privacy definition, and given a relation R, constant k, and a set of diversity constraints Σ, we study the problem of publishing a k-anonymized and diverse instance R^*. An example of a diversity

constraint $\sigma_1 = (\mathsf{ETH}[Asian], 2, 5)$ requires an anonymized instance to contain a minimum of two Asian individuals and no more than five, which is satisfied by Table 5.1 and in Table 5.2. Diversity constraints provide a declarative definition of the minimum and maximum frequency bounds that specific attribute domain values should appear in R^* [7].

In this chapter, we define the (k, Σ)-anonymization problem, which seeks an optimal k-anonymous instance R^* that satisfies a set of diversity constraints, such as $\sigma_1 \in \Sigma$. Note that Σ denotes a set of diversity constraints in this chapter. We study the (k, Σ)-anonymization decision problem, that is, whether there exists a k-anonymous instance R^* that satisfies Σ. In Example 5.1.1, there is no 3-anonymized version of R that satisfies $\sigma_2 = (\mathsf{ETH}[African], 1, 3)$ because there are only two African patients in R.

We propose the *DIVA* algorithm to compute a DIVerse and Anonymized R^*. *DIVA* integrates anonymization with diversity by applying value suppression to find a k-anonymous instance satisfying a set of diversity constraints.

Contributions. We make the following contributions:

1. We define the (k, Σ)-anonymization problem that seeks a k-anonymous relation with value suppression that satisfies Σ. We introduce *DIVA*, a clustering-based algorithm that solves the (k, Σ)-anonymization problem with minimal suppression.
2. We present two selection strategies to improve the *DIVA* algorithm performance by selectively ordering candidate constraints and clusterings to minimize conflict and save computation.
3. We conduct an extensive evaluation using real data collections demonstrating the effectiveness and efficiency of our selection strategies over the naive version of

DIVA and show the utility of diversity constraints over an existing baseline.

Organization. In Section 5.2, we introduce the related work. In Section 5.3, we present necessary definitions and notation. We introduce our algorithm in Section 5.4. We discuss selection strategies for diversity clustering in Section 5.5. We present our evaluation results in Section 5.6, and conclude in Section 5.7.

5.2 Related Work

Privacy Preserving Data Publishing. Extensions of k-anonymity include *l-diversity*, *t-closeness*, *(X,Y)-privacy*, and *(X,Y)-anonymity* with tighter privacy guarantees [66]. *DIVA* is extensible to re-define the clustering criteria according to these privacy semantics. *Differential privacy (DP)* provides a higher level of protection for individuals where the existence (or not) of a single record should not impact the outcome of any statistical analysis [69]. Cuenca et. al, study PPDP in linked data by formalizing the anonymization problem and its complexity for RDF graphs [100]. Hay et. al., present a data publishing algorithm that guarantee anonymity over social network data [101]. In generalization, data values are replaced with less specific, but semantically consistent values according to a generalization hierarchy [66]. While our algorithm currently considers suppression (a special case of generalization), we are exploring distance metrics to include generalization.

Fairness and Diversity. Achieving fair and equal treatment of groups and individuals is difficult in data-driven decision making [102]. Despite a strong need for algorithmic fairness and data diversity, such principles are rarely applied in practice [103]. Data sharing of private data has been studied along two primary lines.

First, causality reasoning aims to recognize discrimination to achieve algorithmic transparency and fairness. Recent techniques have proposed influence measures to identify correlated attributes [104], statistical reasoning about discrimination [105], and reasoning between causality and fairness to generate bias-free, differentially private synthetic data [106]. Secondly, recent work have studied variants of DP to release synthetic data with similar statistical properties to the input data [107], publishing differentially private histograms [108], and studying the impact of differentially private algorithms on equitable resource allocation, especially for strict privacy-loss budgets [109]. Our work is complementary to these efforts, with a different goal; to publish diverse and anonymized versions of the original data with minimal information loss for applications where statistical summaries, synthetic data, and aggregate queries are inadequate. Recent work by Stoyanovich et. al., study diversity in the set selection problem and introduce diversity constraints to guarantee representation for each category in the selected set [110, 7]. We build upon this work, and are the first to formalize diversity constraints and study their foundations. We propose algorithms to couple diversity with data anonymization, a problem not considered in existing work.

Diverse Clustering. Incorporating diversity into clustering has been limited to producing more diverse results. Nguyen et. al., start with an initial clustering and then generate additional clusterings that minimize error from the initial set [111]. Phillips et. al., argue that there is limited success by being too reliant on the initial clustering, and propose a sampling approach to select a diverse, large sample of non-redundant clusters while maximizing a quality metric [112]. The only work we are aware of that combines clustering with anonymization is by Li et. al., that study a

Symbol	Description
$R, \mathcal{R}$	relation and relational schema
X, Y, Z	sets of relational attributes
$\sqsubseteq, \star$	suppression relation, symbol for a suppressed value
σ, Σ	single and set of diversity constraints
$C, \mathcal{C}$	cluster and clustering (set of clusters)

Table 5.4: Summary of notation and symbols.

2-approximation algorithm for l-diversity, an extension of k-anonymity, where each cluster is of size at least l, and each point is a different color (i.e., sensitive value) [113]. However, while our work shares a similar spirit, Li et. al., show that a solution may not be possible depending on the color distribution, and record deletion may be necessary. our algorithm does not consider tuple deletion, and we use graph coloring to model tuple overlap between constraints, focusing instead on a declarative specification of diversity that is realizable in practice.

5.3 Preliminaries

5.3.1 Diversity Constraints

Diversity constraints are originally proposed for the set selection problem defined as follows [7]. Given a set of N items, each associated with a sensitive attribute and a utility score, the *set selection problem* is to select M items to maximize a utility score subject to diversity constraints. The utility score is the sum of scores of each selected item. Let there be d distinct values of the sensitive attribute and m_i with $i \in [1, d]$

be the number of selected items with each distinct value such that $m_i \in [0, M]$ and $\sum_i(m_i) = M$. A diversity constraint σ of the form $floor_i \leq m_i \leq ceiling_i$ specifies upper and lower bounds on m_i, i.e. the number of items with the i-th sensitive value. These constraints ensure representation from each category known as coverage-based diversity. To avoid tokenism, where there is only a single representative from each category, we can increase the lower bound, e.g., $m_i > 1$. Given a set of diversity constraints Σ of the form $\sigma \in \Sigma$, we define our initial problem statement.

Definition 5.3.1 (Problem Statement ((k, Σ)-anonymization))**.** Consider a relation R of schema $\mathcal{R}$, a constant k, a set of diversity constraints Σ. The (k, Σ)-*anonymization problem* is to find a relation R^* where: (1) $R \sqsubseteq R^*$, (2) R^* is k-anonymous, (3) $R^* \models \Sigma$, and (4) R^* has minimal information loss, i.e., a minimum number of $\star$'s.

5.4 The DIVA Algorithm

We present the DIVersity and Anonymization algorithm (*DIVA*) that solves the (k, Σ)-anonymization problem. *DIVA* takes as input a relation R, a minimal and satisfiable set of diversity constraints Σ, constant k, and returns a k-anonymous and diverse relation R' that satisfies Σ. *DIVA* is a clustering-based anonymization algorithm that works in two phases: (i) *clustering*, by partitioning R into disjoint clusters of size $\geq k$; and (ii) *suppression*, by suppressing a minimal number of QI values in each cluster such that they have the same QI values, and form a QI-group of size $\geq k$. The result is a k-anonymous relation, as every QI-group is of size $\geq k$.

Algorithm 10 presents the *DIVA* algorithm details. In the clustering phase in Line 1, *DIVA* uses the *DiverseClustering* procedure to generate a set of diverse clusters $\mathcal{C}_\Sigma$. These clusters guarantee that the diversity constraints in Σ will be satisfied by

Algorithm 10: *DIVA* (R, Σ, k)

Output: k-anonymous and diverse relation.

1 $\mathcal{C}_\Sigma :=$ *DiverseClustering*(R, Σ, k);

2 **if** $\mathcal{C}_\Sigma = \emptyset$ **then return unsatisfiable**;

3 $R_\Sigma :=$ *Suppress*$(\mathcal{C}_\Sigma)$;

4 **foreach** $C_i \in \mathcal{C}_\Sigma$ **do** $R := R \setminus C_i$;

5 $R_k :=$ *Anonymize*(R, k);

6 **return** *Integrate*(R_Σ, R_k);

R_Σ after the suppression phase in Line 3. If there is no k-anonymous relation R' that satisfies Σ, there is no such clustering, and *DIVA* returns $\mathcal{C}_\Sigma := \emptyset$. We provide details of *DiverseClustering* in Section 5.4.1.

In the suppression phase, *DIVA* suppresses values according to the clusters in $\mathcal{C}_\Sigma$. The *Suppress* procedure iterates over tuples in each cluster of $\mathcal{C}$, and suppresses A_i attribute values if there is more than one value for A_i in the same cluster. Assuming each cluster in $\mathcal{C}$ contains at least k tuples, the result of *Suppress* in R is a k-anonymous relation.

Returning to Algorithm 10, *DIVA* anonymizes the remaining tuples of R that are not in $\mathcal{C}_\Sigma$ (Line 4) by applying an existing k-anonymization algorithm (Line 5). *DIVA* is amenable to any k-anonymization algorithm. In Line 6, *Integrate* returns $R' = R_\Sigma \cup R_k$ if $R' \models \Sigma$. Otherwise, R' falsifies the upper bounds of some of the constraints in Σ because of R_k, and *Integrate* resolves this by suppressing minimal values in R' to satisfy Σ.

Example 5.4.1. Consider relation R in Table 5.1, $k = 2$, and $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$, where $\sigma_1 = (\mathsf{ETH}[Asian], 2, 5)$, $\sigma_2 = (\mathsf{ETH}[African], 1, 3)$ and $\sigma_3 = (\mathsf{CTY}[Vancouver], 2, 4)$.

DiverseClustering returns a clustering $\mathcal{C}_\Sigma = \{C_1, C_2, C_3\}$ where $C_1 = \{t_9, t_{10}\}$, $C_2 = \{t_5, t_6\}$, and $C_3 = \{t_7, t_8\}$. Tuples t_9, t_{10} contain the same value $\mathsf{ETH} = Asian$, and together with C_1 guarantee that the lower bound in σ_1 will be satisfied. C_2 and C_3 satisfy the lower bounds of σ_2 and σ_3 for $\mathsf{ETH} = African$ and $\mathsf{CTY} = Vancouver$, respectively. Note that other clusterings, which satisfy Σ, are possible, such as $\{C_2, \{t_8, t_{10}\}\}$. In Section 5.4.1, we describe how we select one of these clusterings.

DiverseClustering returns an empty set if there is no clustering that satisfies Σ. For example, if $k = 3$ there is no possible anonymization that satisfies σ_1, σ_3. In particular, there are no clusters of size 3 that preserve both Vancouver and Asian. For $k = 2$, *DIVA* continues with the *Suppress* procedure that transforms the tuples in $\mathcal{C}_\Sigma$ to $R_\Sigma = \{g_5, ..., g_{10}\}$ as shown in Table 5.3. *DIVA* anonymizes the remaining tuples $R \setminus \mathcal{C}_\Sigma = \{t_1, t_2, t_3, t_4\}$ using an existing k-anonymization algorithm that minimizes the number of $\star$s. In this case, the optimal result is $R_k = \{g_1, g_2, g_3, g_4\}$ in Table 5.3. The *Integrate* procedure returns $R_\Sigma \cup R_k$, which satisfies Σ.

Integrate resolves any inconsistency caused by adding R_k. For example, if $\Sigma = \{\sigma_1, ..., \sigma_4\}$ in which $\sigma_4 = (\mathsf{GEN}[Male], 1, 3)$, $R_\Sigma \cup R_k \not\models \sigma_4$ because there are 4 males in $R_\Sigma \cup R_k$. *Integrate* suppresses GEN in g_5, g_6 or g_3, g_4 to satisfy σ_4. □

5.4.1 Diverse Clustering

We now describe the *DiverseClustering* routine in the *DIVA* algorithm, and define a clustering that satisfies a diversity constraint.

Definition 5.4.1. Given a diversity constraint σ over a relation R and a clustering $\mathcal{C}$ with clusters of tuples in R, $\mathcal{C}$ satisfies σ, denoted as $\mathcal{C} \models \sigma$ if $Suppress(\mathcal{C}) \models \sigma$. The clustering $\mathcal{C}$ satisfies a set of constraints Σ, if $\mathcal{C} \models \sigma_i$ for every $\sigma_i \in \Sigma$.

In Example 5.4.1, $\mathcal{C} = \{C_1\}$ satisfies σ_1 since *Suppress*($\mathcal{C}$) = $\{g_9, g_{10}\}$ (cf. Table 5.3) satisfies σ_1. The objective of *DiverseClustering* is to find $\mathcal{C}_\Sigma$ that satisfies Σ. This works by computing clustering $\mathcal{C}_{\sigma_i}$ that satisfy diversity constraints $\sigma_i \in \Sigma$, and then computing $\mathcal{C}_\Sigma$ by merging the clusterings $\mathcal{C}_{\sigma_i}$. The main challenge is to ensure the clustering for each σ_i is consistent with clusterings for the other constraints in Σ. If so, this allows us to merge the $\mathcal{C}_{\sigma_i}$ to obtain $\mathcal{C}_\Sigma$.

Definition 5.4.2 (Consistent clusterings)**.** Consider diversity constraints σ_i and σ_j over relation R. Two clusterings $\mathcal{C}_{\sigma_i}$ and $\mathcal{C}_{\sigma_j}$ are consistent if and only if $\mathcal{C}_{\sigma_i} \models \sigma_i$ and $\mathcal{C}_{\sigma_j} \models \sigma_j$ implies *Merge*($\mathcal{C}_{\sigma_i}, \mathcal{C}_{\sigma_j}$) $\models \{\sigma_i, \sigma_j\}$.

Merge in Defn. 5.4.2 merges clusters if they overlap, otherwise their union is computed, e.g., *Merge*($\{\{t_5, t_6\}\}, \{\{t_6, t_7\}\}$) = $\{\{t_5, t_6, t_7\}\}$, and *Merge*($\{\{t_5, t_6\}\}, \{\{t_7, t_8\}\}$) = $\{\{t_5, t_6\}, \{t_7, t_8\}\}$. We can check the consistency of two clusterings using *Merge* and *Suppress*.

Example 5.4.2. In Example 5.4.1, $\mathcal{C}_2 = \{\{t_5, t_6\}\}$ and $\mathcal{C}_3 = \{\{t_6, t_7\}\}$ are not consistent w.r.t σ_2 and σ_3, because $\mathcal{C}_2 \models \sigma_2$ and $\mathcal{C}_3 \models \sigma_3$, but *Merge*($\mathcal{C}_2, \mathcal{C}_3$) = $\{\{t_5, t_6, t_7\}\} \not\models \{\sigma_2, \sigma_3\}$. This occurs since t_6 appears in two different clusters $\{t_5, t_6\}$ and $\{t_6, t_7\}$ in $\mathcal{C}_2$ and $\mathcal{C}_3$, respectively. Consequently, the value *Vancouver* will be suppressed in the clustering *Merge*($\mathcal{C}_2, \mathcal{C}_3$) because $t_6[\mathsf{CTY}] \neq t_5[\mathsf{CTY}]$, and hence, σ_3 will not be satisfied. □

It is straightforward to show that if $\mathcal{C}_{\sigma_i} \models \sigma_i$ for every $\sigma_i \in \Sigma$, and every pair of $\mathcal{C}_{\sigma_i}, \mathcal{C}_{\sigma_j}$ are consistent, we can generate $\mathcal{C}_\Sigma \models \Sigma$ by merging all clusterings $\mathcal{C}_{\sigma_i}$. Note that it is not necessary to check consistency of every pair of clusterings $\mathcal{C}_{\sigma_i}, \mathcal{C}_{\sigma_j}$, as we only need to check if σ_i, σ_j apply to some tuples that are common to both constraints.

We use this intuition to transform our problem of computing all $\mathcal{C}_{\sigma_i}$ to the problem of graph coloring.

We model the problem of searching for the clusterings $\mathcal{C}_{\sigma_i}$ as a graph coloring problem. Given an undirected graph $G = (\Gamma, E)$, where Γ and E denote the set of vertices and edges, respectively, and m distinct colors, the graph coloring problem is to color all vertices subject to certain constraints. In its simplest form, no two adjacent vertices can have the same color.

For relation R and diversity constraints Σ, we model each diversity constraint $\sigma_i \in \Sigma$ as a vertex $v_i \in \Gamma$. We use $v_i.constraint$ to refer to σ_i. We define the *relevant tuples* of σ_i, denoted $I_{\sigma_i} \subseteq R$, as tuples containing the target values of σ_i. We record the relevant tuples of σ_i in vertex v_i. An edge $e_{ij} \in E, e_{ij} = (v_i, v_j)$, exists between vertices v_i and v_j if there is at least one tuple in the intersection of their relevant tuple sets, i.e., $(I_{\sigma_i} \cap I_{\sigma_j}) \neq \emptyset$. In Example 5.4.1, G contains three vertices corresponding to $\sigma_1, \sigma_2, \sigma_3$ (cf. Figure 5.1), and two edges $E = \{(v_1, v_3), (v_2, v_3\}\}$. The relevant sets $I_{\sigma_1} = \{t_8, t_9, t_{10}\}$, $I_{\sigma_3} = \{t_6, t_7, t_8, t_{10}\}$, have a non-empty intersection of $\{t_8, t_{10}\}$. Similarly, for $I_{\sigma_2} = \{t_5, t_6\}$, $I_{\sigma_2} \cap I_{\sigma_3} = \{t_6\}$. We note that $I_{\sigma_1} \cap I_{\sigma_2} = \emptyset$. Choosing a color c_i for vertex v_i is analogous to finding a clustering $\mathcal{C}_{\sigma_i}$ for σ_i. In our setting, to color two adjacent v_i, v_j, we must check that their clusterings $\mathcal{C}_{\sigma_i}$ and $\mathcal{C}_{\sigma_j}$ are consistent. We define $c_i.clustering$ to refer to clustering corresponding to color c_i.

Algorithm 11 presents the details of *DiverseClustering*. We build the graph G for Σ and R (Line 1). We then initialize the clustering $\mathcal{C}_\Sigma$ and a mapping V that stores the color (assigned clustering) for each vertex (Line 2), and checks if a coloring exists via *Coloring*.

Algorithm 12 presents the recursive function, *Coloring*, that takes a graph G, the

Algorithm 11: *DiverseClustering*(R, Σ, k)

Output: Clustering $\mathcal{C}_\Sigma$.

1 $G :=$ *BuildGraph*(R, Σ);

2 $V := \emptyset; \mathcal{C}_\Sigma := \emptyset$;

3 **if** *Coloring*(G, V, R) **then**

4 $\quad$ **foreach** $\langle v_i, c_i \rangle \in V$ **do** $\mathcal{C}_\Sigma :=$ *Merge*$(\mathcal{C}_\Sigma, c_i.clustering)$;

5 **return** $\mathcal{C}_\Sigma$;

mapping V (specifying the colored vertices), relation R, and returns *true* if the remaining vertices of G can be colored; otherwise it returns *false*. In the naive version, *Coloring* randomly selects an uncolored vertex (Line 2) to color using *NextVertex*. In Section 5.5, we present two strategies for selecting candidate vertices. Given a vertex v, we try to color v by checking whether the candidate clustering of v and its adjacent vertices are inconsistent (Lines 3-12). The routine *Clusterings* returns candidate clusterings $\mathcal{C}$ that satisfy $v.constraint$ ($Suppress(\mathcal{C}) \models v.constraint$). For example, in Example 5.4.1, $Clusterings(\sigma_1, R)$ contains four different clusterings $\{\{t_8, t_9\}\}$, $\{\{t_8, t_{10}\}\}$, $\{\{t_9, t_{10}\}\}$, $\{\{t_8,\ t_9, t_{10}\}\}$, while $Clusterings(\sigma_2, R)$ contains only one clustering $\{\{t_5,\ t_6\}\}$. In the naive algorithm, we assume *Clusterings* returns clusterings in random order. We present strategies in Section 5.5 to order the clusterings to minimize inconsistencies. In Lines 4-12, we check whether $\mathcal{C}$ has inconsistency with the clustering of any constraint modeled by an adjacent vertex v' to v. If they are consistent, we generate a new color c assigned to the clustering $\mathcal{C}$, and we temporarily color v with c by adding $\langle v, c \rangle$ to V. We then recursively call *Coloring* to check whether the remaining vertices in G can be colored. If the color c does not work, i.e. *Coloring* returns false in Line 11, we remove $\langle v, c \rangle$ from V, and try another color.

Algorithm 12: *Coloring*(G, V, R)

Output: **true** if there exists a coloring of G, otherwise **false**.

```
1  if V contains all vertices of G then return true;
2  v := NextVertex(G, V);
3  foreach C ∈ Clusterings(v.constraint, R) do
4      consistent := true;
5      foreach ⟨v', c'⟩ ∈ V s.t. v' is adjacent to v do
6          if C and c'.clustering are inconsistent then
7              consistent := false; break;
8      if consistent then
9          c := new color with clustering C;
10         V := V ∪ {⟨v, c⟩};
11         if Coloring(G, V, R) then return true ;
12         V := V \ {⟨v, c⟩};
13 return false
```

If all clusterings are inconsistent, i.e., there is no successful coloring of v, we return false in Line 13, to backtrack and evaluate a different vertex.

Example 5.4.3. Consider an execution of Alg. 12 *Coloring* on the graph G in Figure 5.1, with vertices $\{v_1, v_2, v_3\}$ representing constraints $\{\sigma_1, \sigma_2, \sigma_3\}$, respectively. The candidate clusterings that satisfy each constraint (i.e., the output of the routine *Clusterings*) are shown beside each vertex. Consider vertex v_1 first (Line 2), and we select $S_{\sigma_1} = \{t_9, t_{10}\}$, which is consistent with any other clustering. We then try to color vertices v_2 and v_3 by recursively calling *Coloring* in Line 11. If vertex v_2 is

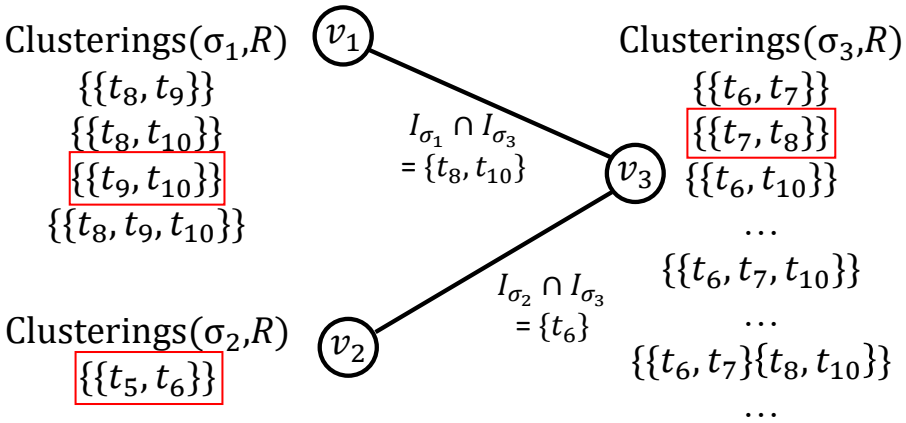


Figure 5.1: Diverse clustering as graph coloring.

selected, the only clustering is $\mathcal{C}_{\sigma_2} = \{\{t_5, t_6\}\}$ that is consistent with $\mathcal{C}_{\sigma_1}$. Considering the last vertex v_3, we iterate over the clusterings for σ_3, and determine that the only consistent clustering (w.r.t. $\mathcal{C}_{\sigma_1}$ and $\mathcal{C}_{\sigma_2}$) is $\{\{t_7, t_8\}\}$, which we assign to $\mathcal{C}_{\sigma_3}$. Since we have found a clustering satisfying all constraints (i.e., a coloring of all vertices), the *Coloring* routine returns *true* with V containing the vertices and their colors (i.e., clusterings). The calling routine *DiverseClustering* uses V to compute the final clustering as $\mathcal{C}_\Sigma = \{\{t_5, t_6\}, \{t_7, t_8\}, \{t_9, t_{10}\}\}$. □

Runtime Analysis. *DIVA* runs in polynomial time w.r.t. the number of constraints since *DiverseClustering*, *Anonymize*, and *Suppress* run in polynomial time. In particular, the size of these clusters is in $[k, 2k-1]$ (where k is a parameter from k-anonymity) and there are polynomially many clusters of each size. Note that there is no cluster of size $\geq 2k$ because we can split them into clusters of size $\geq k$. *DIVA* runs in polynomial time w.r.t. the size of $|R|$ (the number of tuples). This runtime depends on *Coloring* where we consider $O(N^{2k-1})$ clusters of size in $[k, 2k-1]$ as k-choose-N, (k+1)-choose-N, ..., and (2k-1)-choose-N are all in $O(N^{2k-1})$. In the next section, we present strategies to improve the performance of *Coloring* while

evaluating the space of possible assignments.

5.5 Selection Strategies

In the naive version of *DIVA*, we randomly select a constraint and a clustering to evaluate. These choices impact algorithm performance as poor initial selections can lead to increased backtracking operations downstream. We selectively order the constraints (vertices) and clusterings (colors) that most likely lead to a graph coloring while minimizing the need to backtrack. We start evaluating constraints (vertices) that are the most difficult to satisfy. By postponing these candidates, we may encounter fewer or no possible consistent clusterings as we assign clusterings to less restrictive constraints. We apply this intuition to propose the following two strategies.

DIVA-MinChoice: Our preference is to select constraints with the fewest candidate clusterings, as we start with the most restrictive constraints first, i.e., those with the fewest choices, ensuring that these constraints are first satisfied. In the routine *NextVertex*, we initially select a vertex v with a minimum value $|Clusterings(v.constraint, R)|$. As we visit vertices and assign (colors) clusterings, we update the candidate clusterings for their neighbors.

DIVA-MaxFanOut: In this strategy, we target constraints that overlap with the highest number of other constraints. This is modeled in the graph G as vertices with the maximum number of unvisited edges. We preferentially select these constraints due to their high number of interactions with other constraints, which lead to an increased number of target attributes, and bounds that the relevant tuples must satisfy. This heuristic strategy aims to satisfy "maximum overlap" constraints first, and

perform early pruning of unsatisfiable clusterings to reduce the number of clustering evaluations downstream. The vertex selection in this strategy is similar to *incidence degree ordering* in graph coloring [114].

In both strategies, *Clusterings* returns a list of clusterings in ascending order of the number of overlapping tuples. For instance, for a clustering $\mathcal{C}$ and a neighboring vertex v (constraint σ), overlapping tuples are in the target I_σ and in a cluster in $\mathcal{C}$. In Section 5.6.4, we show these strategies improve runtime by an average 24%.

Example 5.5.1. In Fig. 5.1, the DIVA-MinChoice strategy first selects vertex v_2 (σ_2), since $|\mathit{Clusterings}(\sigma_1, R)| = 4$, $|\mathit{Clusterings}(\sigma_2, R)| = 1$, $|\mathit{Clusterings}(\sigma_3, R)| = 12$. After assigning cluster $\{\{t_5, t_6\}\}$ to v_2, we update the clusterings, and vertices v_1, v_3 will each have 4 clusterings; we break ties randomly. In DIVA-MaxFanOut, we first select vertex v_3 (σ_3) containing two unvisited edges. *Clusterings* then computes cluster $\{\{t_7, t_8\}\}$ has 2 overlapping tuples (t_8, t_{10} are in I_{σ_1}). Similarly, cluster $\{\{t_7, t_8\}\}$ has 1 overlapping tuple t_8 in I_{σ_1}. Hence, clustering $\{\{t_7, t_8\}\}$ is ranked first assuming it wins the tie against clustering $\{\{t_7, t_{10}\}\}$. We randomly select between v_1 and v_2 given their equal number of unvisited edges. □

5.6 Experiments

Our evaluation has the following objectives: (1) We evaluate *DIVA*'s accuracy using three types of diversity constraints as we vary k, and the conflict rate among tuples. (2) We evaluate the accuracy and performance of all *DIVA* variants as we vary $k, |\Sigma|$, the conflict rate, and the target attribute(s) data distribution. (3) We compare against an existing k-anonymization baseline algorithm to evaluate the cost of introducing diversity constraints into data anonymization.

	Pantheon	**Census**	**Credit**	**Population (Syn)**
$\|R\|$	11,341	299,285	1000	100,000
n	17	40	20	7
$\|\Pi_{QI}(R)\|$	5,636	12,405	60	24,630
$\|\Sigma\|$	24	21	18	10

Table 5.5: Data characteristics.

5.6.1 Experimental Setup

We implement *DIVA* using Python 3.6 on a server with 32 Core Intel Xeon 2.2 GHz processor with 32GB RAM. We describe the datasets, diversity constraints, and baseline comparative algorithm.

Datasets. We use three real data collections and one synthetic dataset. Table 5.5 gives the data characteristics, showing a range of data sizes w.r.t. the number of tuples ($|R|$), number of attributes (n), number of unique values in the QI attributes ($|\Pi_{QI}(R)|$), and the total number of defined diversity constraints ($|\Sigma|$).

Pantheon [115]. This dataset describes individuals based on the popularity of their biographical page in Wikipedia. Attributes include name, sex, city, country, continent. We select sex, city, country and continent as QI attributes, and define diversity constraints on sex and continent, where the attribute domain is two and six, respectively. We use this dataset to evaluate algorithm accuracy.

Census [116]. The U.S. Census Bureau describes population data for 1970, 1980 and 1990. We select sex, workclass, marital status, family relationship, race, and native country as QI attributes. We define (single and multi-attribute) diversity constraints on the sex and race attribute domains with size two and five, respectively.

We evaluate accuracy, runtime, and comparative performance with this dataset.

German Credit [116]. This dataset classifies persons as good or bad credit risk according to attributes such as credit history, credit amount, sex, job, housing, marital status, and stratified savings account balances. We select sex, job, housing, saving account as QI attributes, and define diversity constraints on sex and job containing two and four values, respectively. We comparatively evaluate against an existing k-anonymization baseline with this dataset.

Synthetic Population Data (Pop-Syn). We use the Synner.io tool to generate realistic synthetic data by declaratively specifying the desirable distribution properties in the target attributes [117]. We generate a synthetic dataset describing population characteristics (age, education, race, gender, income, marital status, occupation). We select a subset of these attributes as target attributes, and vary their statistical distributions (uniform, Gaussian, Zipfian) to study the impact on *DIVA*'s accuracy.

Diversity Constraints. We implement different notions of diversity such as minimum frequency, average and proportional representation from the attribute domain. We use the diversity definitions presented by Stoyanovich et. al. that define three classes of diversity constraints as described below [7]. We generate a set of satisfiable diversity constraints Σ_i for each class, $i = \{1, 2, 3\}$, for each dataset.

In the original definition, Stoyanovich et. al., define these diversity constraint classes w.r.t. the number of selected elements from a set [7]. In our setting, we consider an equivalent notion as the number U of published (non-suppressed) tuples in R'. To estimate U, recall the tuples in the QI attributes are suppressed to achieve the indistinguishability of a tuple among $(k-1)$ other tuples in a cluster group. We can estimate U by computing the cardinality of the QI attribute(s) domain, and

subtracting this value from the size of R. Let $\Pi_{QI}(R)$ represent the projection of relation R on the QI attributes, i.e., the set of unique tuples w.r.t. the QI attributes. These unique values will need to be suppressed among an average of $\frac{|R|}{k}$ groups to achieve k-anonymity. Hence, we estimate $U = |R| - |\Pi_{QI}(R)|$ as the number of tuples that are published (unsuppressed) tuples in R'. We now describe each class of diversity constraints. Let $d = |dom(A)|$, i.e., the number of unique values in the target attribute(s) A domain. The full set of diversity constraints, datasets and our code are available at [118].

- **Minimum:** Cover as many values in the attribute(s) domain as possible. If $U > d$, set $\lambda_l = \lambda_r = 1$ for all d (value) constraints. Then, compute $w = U - d$. If $w > 0$, then assign these values to a random constraint σ'_j by setting its $\lambda'_r = \lambda'_r + w$. Select σ'_j randomly where $freq(a) \geq \lambda'_r + w$.

  If $U < d$, set $\lambda_l = \lambda_r = 1$ to a random set of U out of d constraints, and set $\lambda_l = \lambda_r = 0$ to the remaining $d - U$ constraints.
- **Average:** Select equal numbers for each value in the attribute domain. If $U \geq d$, set $\lambda_l = min(\lfloor U/d \rfloor, freq(a))$, $\lambda_r = min(\lceil U/d \rceil, freq(a))$, where $freq(a)$ represents the frequency of value(s) a in attribute(s) A in R. Next, compute $w = \sum_{i=1}^{d} \lambda_{r_i}$. If $w < U$, then assign these values to a random σ'_j by setting $\lambda'_r = \lambda'_r + w$. Select σ'_j randomly where $freq(a) \geq \lambda'_r + w$. If $U < d$, define as in **minimum** class.
- **Proportion:** Select equal proportions for each value in $dom(A)$. If $U \geq d$, set $\lambda_l = \lfloor U * freq(a)/|R| \rfloor, \lambda_r = \lceil U * freq(a)/|R| \rceil$. If $U < d$, set constraints as in **minimum** class above.

Comparative Baseline. *DIVA* runs in polynomial time w.r.t. $|R|$ since *DiverseClustering*, *Anonymize*, and *Suppress* run in polynomial time. *DiverseClustering*

Symbol	**Description**	**Values**
$\|R\|$	#tuples	60k, 120k, 180k, 240k, 300k
$\|\Sigma\|$	#constraints	4, **8**, 12, 16, 20
$cf(\Sigma)$	conflict rate	0, **0.2**, 0.4, 0.6, 0.8, 1
k	minimum cluster size	10, **20**, 30, 40, 50

Table 5.6: Parameter values (defaults in bold)

Method	norm disc
MinChoice [118]	0.62
MaxFanOut [118]	0.51
k-member [119]	0.34

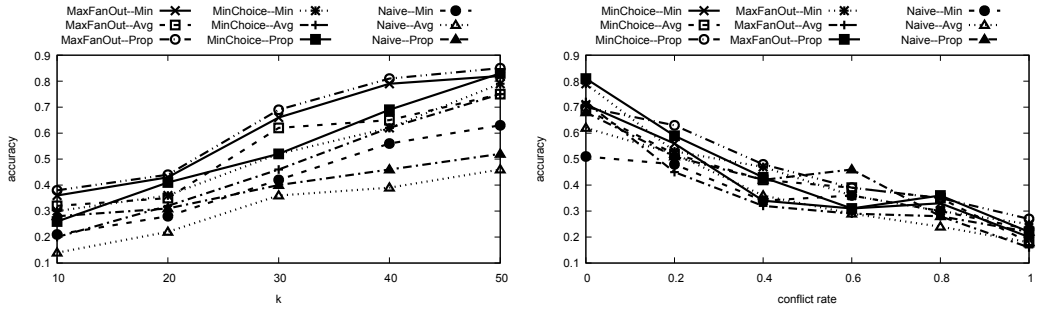
Table 5.7: Comparison Summary

and its recursive procedure *Coloring* run in polynomial time w.r.t. $|R|$ since the number of candidate clusterings for each constraint is polynomial w.r.t. $|R|$. In particular, the size of these clusters is in $[k, 2k-1]$ and there are polynomially many clusters of each size. Note that there is no cluster of size $\geq 2k$ because we can split them into clusters of size $\geq k$. *DiverseClustering* and *DIVA* run in exponential time w.r.t. $|\Sigma|$ since we can assign $O(|R|)$ different clusterings to each constraint. In the next section, we present strategies to improve the performance of *Coloring* while evaluating the space of possible assignments.

5.6.2 Metrics and Parameters

Metrics. We compute the average runtime over five executions. To quantify accuracy, we use an intuitive measure to model desirable anonymizations that minimize a cost function. Existing anonymization algorithms use cost functions that minimize information loss from suppression [66]. The resulting anonymized relation R' can be considered as imposing a penalty on each tuple that reflects its information loss due to suppression. The *discernibility metric*, $disc(R', k)$, quantifies the differentation between tuples for a given k value, by assigning a penalty to each tuple based on the number of tuples that are indistinguishable from it in R' [120]. If an unsuppressed tuple lies in a cluster of size j, it is assigned a penalty of j. If a tuple is suppressed, it is assigned a penalty of $|R'|$ since the tuple cannot be differentiated from other tuples in R' [120]. We define the normalized discernibility score as $\widehat{disc}(R', k) = \frac{disc(R',k)}{|R'^2|}$. To quantify accuracy, we compare $disc(R', k)$ for R' computed by *DIVA* against $disc(R'', k)$ for the best R'' computed by sampling among all the possible clusters and selecting the best clustering. We compute accuracy as the ratio of the normalized discernibility scores $\frac{\widehat{disc}(R'',k)}{\widehat{disc}(R',k)}$. In our comparative evaluation, we measure accuracy using $\widehat{disc}(R', k)$ to quantify the penalty to enforce diversity in R'.

Parameters. Unless otherwise stated, Table 5.6 shows the range of parameter values we use, with default values in bold. We measure the *conflict rate* between the diversity constraints by measuring the number of overlapping relevant tuples between a pair of diversity constraints. We use Jaccard similarity to quantify the similarity between two sets, computed as the size of the intersection divided by the size of the union of the sets. Similarly, we define the conflict rate $cf(\sigma_i, \sigma_j) = \frac{|I_{\sigma_i} \cap I_{\sigma_j}|}{|I_{\sigma_i} \cup I_{\sigma_j}|}$ between constraints σ_i, σ_j, and I_{σ_i} refers to the relevant tuples of σ_i. For all $\sigma_i \in \Sigma$, we



(a) Varying Σ, k (Census) (b) Varying Σ, cf (Census)

Figure 5.2: Varying Σ, k, cf.

compute $cf(\Sigma) = \frac{\Sigma_{i=1}^{|\Sigma|} cf(\sigma_i, \sigma_{i+1})}{\binom{|\Sigma|}{2}}$, i.e., the average of all conflict scores for every pair of diversity constraints. Values of $cf(\Sigma)$ range from [0, 1], where 0 indicates no overlapping relevant tuples, and 1 indicates full overlap (exact similarity) of the relevant tuples among the constraints.

5.6.3 Accuracy

We evaluate accuracy using three classes of constraints, and then vary $|\Sigma|$, cf, and the data distribution in the target attribute values.

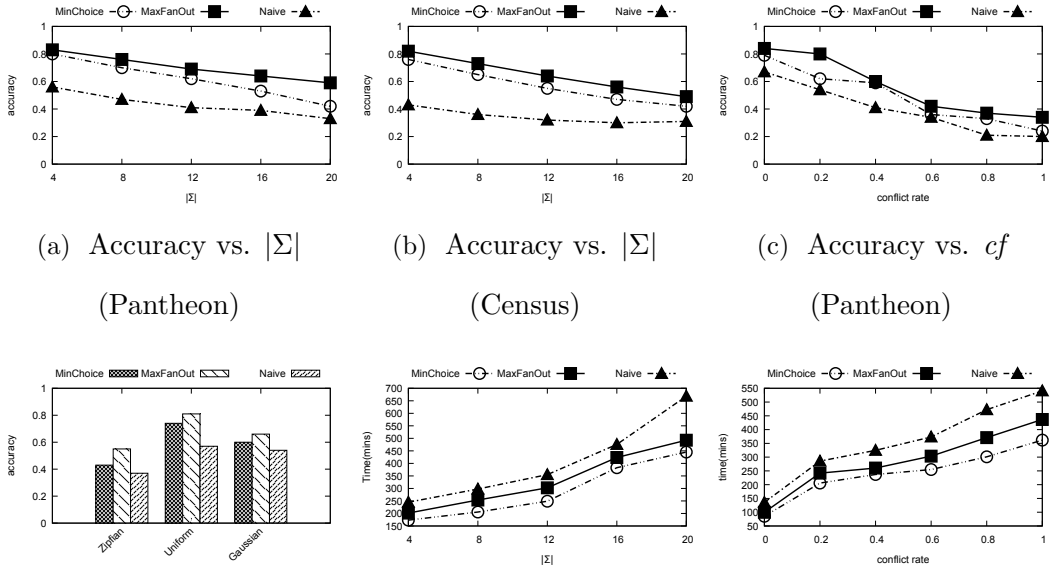
Exp-1: Vary Σ and k. Figure 5.2a gives the *DIVA* accuracy for the three variations of *DIVA* as we vary k using the Census dataset, across the three diversity constraint classes. Accuracy increases for larger k values as more values are suppressed to achieve anonymization. As expected, DIVA-Naive leads to the lowest accuracy due to its random selections. DIVA-MaxFanOut outperforms DIVA-MinChoice by an average +9%, since by ordering clusterings in ascending order according to the number of overlapping tuples, we select clusterings that satisfy a maximal number of dependent

constraints. In contrast, DIVA-MinChoice does not consider this constraint interaction. The proportion class of constraints achieves the best tradeoff between accuracy and adapting to the relative frequency of values in the data. Although the minimum class of constraints achieves higher accuracy in some cases (given the minimal lower bound values), this can lead to tokenization in R'.

Exp-2: Vary Σ and Conflict Rate. Figure 5.2b shows *DIVA* accuracy as we vary the conflict rate (*cf*) across the three constraint classes. Accuracy declines for increasing *cf* as it is more difficult to find a clustering. Again, DIVA-Naive achieves the lowest accuracy, whereas DIVA-MaxFanOut performs best by first selecting clusterings that satisfy neighboring vertices (constraints). The proportion class of constraints capture the relative distribution in the attribute domain (with less sensitivity than average), and avoids tokenization (a drawback of minimum constraints). Henceforth, we run subsequent experiments using the proportion class constraints.

Exp-3: Vary $|\Sigma|$. Figure 5.3a and Figure 5.3b show the *DIVA* accuracy as we vary the number of (proportion) constraints $|\Sigma|$ using the Pantheon and Census dataset, respectively. DIVA-MaxFanOut outperforms DIVA-Naive and DIVA-MinChoice by +27% and +9%, respectively, (Pantheon), and +30% and +7% (Census). As $|\Sigma|$ increases, we see accuracy decline but at a relatively slow linear rate. As a new constraint $\sigma \notin \Sigma$ is added, we observe new relevant tuples w.r.t. σ join existing clusters of relevant tuples from Σ leading to a smaller decline in accuracy. This occurs with multi-attribute constraints that share target attributes with single attribute constraints. The alignment of QI and target attribute values between new and existing tuples influence the accuracy rate of decline.

Exp-4: Vary Conflict Rate. Figure 5.3c shows the *DIVA* accuracy as we vary



(a) Accuracy vs. $|\Sigma|$ (Pantheon)

(b) Accuracy vs. $|\Sigma|$ (Census)

(c) Accuracy vs. *cf* (Pantheon)

(d) Accuracy vs. distrib. (Pop-Syn)

(e) Runtime vs. $|\Sigma|$ (Census)

(f) Runtime vs. *cf* (Census)

Figure 5.3: *DIVA* effectiveness and efficiency.

the conflict rate (*cf*). As expected, accuracy declines for increasing *cf*, with DIVA-MaxFanOut and DIVA-MinChoice outperforming DIVA-Naive by +17% and +9%, respectively. DIVA-MaxFanOut shows improved accuracy over DIVA-MinChoice since targeting constraints with a high number of interactions (with other constraints) first allows it to eliminate unsatisfying clusterings sooner, while also satisfying dependent diversity constraints.

Exp-5: Vary Data Distribution. We generate target attribute values according to the Zipfian, uniform, and Gaussian distributions in the Pop-Syn dataset with $|R| = 100k$ and $|\Sigma| = 8$. Figure 5.3d shows that DIVA-MaxFanOut performs best across all distributions by 8% and 17% over DIVA-MinChoice and DIVA-Naive, respectively. The target uniform distribution performs best as domain values are spread evenly

across the tuples, avoiding contention among a small set of tuples. This conflict occurs more often in the Zipfian case than the Gaussian, leading to lower accuracy.

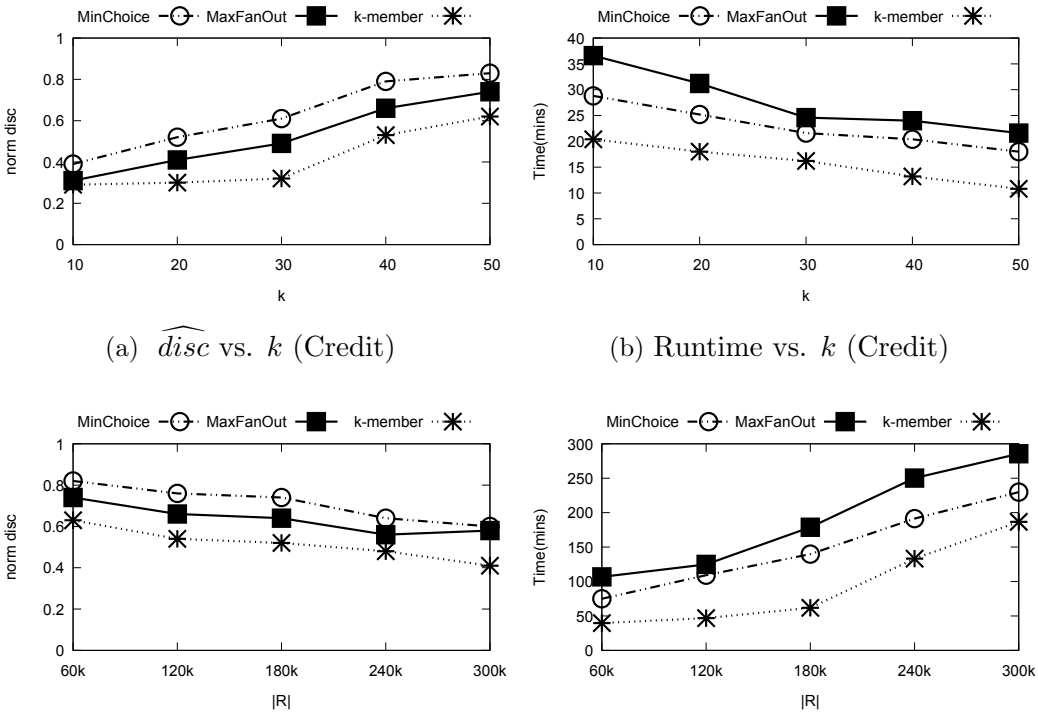
5.6.4 Performance

Exp-6: Scale $|\Sigma|$. Figure 5.3e shows the *DIVA* runtime as we vary the number of constraints over the Census dataset. As expected, DIVA-Naive shows exponential growth for increasing $|\Sigma|$ since we can assign $O(|R|)$ different clusterings to each constraint. Our selection strategies to restrict clusterings and perform early pruning in DIVA-MinChoice and DIVA-MaxFanOut show linear scale-up with a 29% and 18%, respectively, reduction in runtime over the naive version.

Exp-7: Vary Conflict Rate. Figure 5.3f shows runtimes as we vary the conflict rate. DIVA-MinChoice outperforms DIVA-MaxFanOut and DIVA-Naive by 16% and 23%, respectively. We observe that when conflicts occurs among a set of tuples, leaving residual tuples that are unique and the only ones that can satisfy a constraint, e.g., vertex v_2 (σ_2) in Figure 5.1, DIVA-MinChoice performs well. By selecting these special constraints first (with fewer clustering choices), we reduce the number of clusterings to evaluate.

5.6.5 Overhead of Diversity Constraints

Exp-8: Vary k. Figure 5.4a and Figure 5.4b show the comparative discernibility scores and runtimes between *DIVA* and k-member [119]. DIVA-MinChoice and DIVA-MaxFanOut incur an average 32% and 44% higher runtime, respectively, than k-member, reflecting the cost of computing a *diverse* data instance. As k increases, we expect more tuples to be suppressed leading to higher penalty costs, and higher



(a) $\widehat{disc}$ vs. k (Credit)

(b) Runtime vs. k (Credit)

(c) $\widehat{disc}$ vs. $|R|$ (Census)

(d) Runtime vs. $|R|$ (Census)

Figure 5.4: Comparative evaluation.

$\widehat{disc}(R', k)$ scores. For DIVA-MaxFanOut and DIVA-MinChoice, a 10% reduction in $\widehat{disc}(R', k)$ costs 13m and 9m, respectively, whereas for k-member, the cost is 4m. We believe that the overhead and trade-off are still acceptable in practice since constraint validation and anonymization is often done offline. As next steps, we are exploring techniques to reduce the overhead via parallel processing of the *Coloring* routine on subgraphs of G.

Exp-9: Vary $|R|$. Figure 5.4c shows that as $|R|$ increases, $\widehat{disc}(R', k)$ scores slightly improve as QI and target attribute values from the new tuples align with existing tuples, and do not incur additional suppression (penalty). In contrast, when new attribute values are suppressed to satisfy diversity constraints (at $|R| = 240K$), we

incur increased penalty costs. Figure 5.4d shows that *DIVA* runtimes increase linearly w.r.t $|R|$ with an average overhead of 36% over the baseline, as new tuples and clusterings need to be evaluated.

5.7 Conclusion

In this Chapter, we studied the (k, Σ)-anonymization problem and introduce *DIVA*, a DIVersity-driven Anonymization algorithm that computes a privatized data instance guaranteed to satisfy a set of diversity constraints. We evaluated the efficiency and effectiveness of our algorithm with real datasets. The results of evaluation showed that our algorithm can guarantee the data diversity in the anonymization with acceptable overhead. As future work, we intend to study more expressive statistical-based diversity constraints, and privacy extensions beyond k-anonymity. We are also investigating a distributed version of the coloring algorithm in *DiverseClustering* for improved scalability.

Chapter 6

Conclusion and Future Work

In this thesis, we aim to address the challenges of data duplication, data privacy and cleaning, and data diversity. We summarize our contributions and propose directions for future work.

6.1 Data Deduplication

In our first contribution, we present a semantic-aware data deduplication framework, which combines constraint and non-constraint attribute features to identify duplicates. We propose a set of statistical features to capture attribute and constraint properties. We also propose a weighted frequency metric, which can assign greater weight on terms that are better indicators of duplicates. We evaluate our techniques on real datasets and the results show that our techniques achieve improved runtime performance against existing learning-based models, while achieving comparative accuracy.

Future Work. As next steps, we intend to improve the robustness of our framework against missing values by developing imputation methods that impute the missing values. We will experiment with more use cases to refine our duplication metrics to handle greater heterogeneity of data types, and defining duplication scores at the record level. We plan to explore improved matching accuracy by relaxing the strict equality conditions on numeric values to allow matching to include a permitted tolerance window between two values.

6.2 Data Privacy and Data Cleaning

In our second contribution, we presented a privacy-aware data cleaning framework, which preserves (X,Y,L)-anonymity in sensitive data during data cleaning. We introduce generalized databases and queries, along with an extended data pricing scheme that allows the client to securely query the service provider. We propose *SafePrice*, a data pricing algorithm that enforces (X,Y,L)-anonymity in our framework that resolves inconsistencies in the client and uses the notion of generalized repair values to protect the privacy of the service provider.

Future Work. We plan to extend our framework to include multiple service providers and allow price negotiation between client and service providers. The current framework only allows the client to query and purchase from one service provider, and there is no room for bargaining. When multiple providers participate, they may compete with each other through price strategies. Since there are multiple choices, the client can optimize his budget by purchasing data from different service providers to reduce the cost.

Another direction is to include multiple clients in our framework. Once multiple clients are involved, the challenge is to design a mechanism to protect privacy when the clients may exchange the purchased data with each other. An initial idea is to maintain a privacy-revealing table inside the service provider. This privacy-revealing table will keep track of the revealed values from the service provider. Once the privacy-revealing table detects the revealed information has reached the service provider's privacy budget, the service provider will stop answering further queries.

We also plan to extend our framework to other privacy models such as l-diversity for a stronger privacy guarantee. The current privacy model in our framework is based on k-anonymity. It requires that the number of tuples in each QI group should be no less than k to break the linkage between QI attributes and sensitive attributes. If the sensitive values in the same QI group are all equal, then this model will not work. To achieve l-diversity, we need to extend our model to ensure there are at least l different values in the sensitive attributes for each QI group. Since the privacy is determined by two factors: the number of tuples in each QI group and the number of distinct values, we need to adjust our pricing mechanism as well. In that case, our pricing mechanism should consider the impact of the distribution of revealed values as well as the distribution of the remaining values during the query answering process.

6.3 Data Diversity

Finally, we studied the problem of diversifying anonymized data using diversity constraints. We present a diversity-driven anonymization algorithm that anonymizes a given data instance, while ensuring a set of diversity constraints are satisfied.

Future Work. We plan to include data fairness into our framework. The goal

of data fairness is to ensure the results obtained using computerized processing are unbiased by nature. There are two types of data fairness: individual fairness and group fairness [121]. Individual fairness states that two individuals who are similar w.r.t. a particular classification task should be classified similarly. Group fairness states that the proportion of members in a specific group who are classified positively should be statistically indistinguishable from the proportion of members of the overall population. The challenge is to model data fairness and include it into our framework.

We also intend to extend the privacy model in our framework. We plan to include differential privacy(DP) into our framework and study how to satisfy both DP and a set of diversity constraints. The challenge is to quantify the randomization that is introduced by DP and adjust the lower and upper bound of our diversity constraint to adapt to this randomization. A key challenge to exploring DP with diversity constraints, will be to study models that can capture the diversity requirements as part of aggregate queries.

Lastly, we will investigate a distributed version of the diversity clustering algorithm for improved scalability. We will modify our graph colouring algorithm to run in parallel by partitioning the graph into subgraphs. The colouring algorithm will run on these subgraphs in parallel to find the clusterings. Once the colouring algorithm is completed in each subgraph, we can merge the clusterings.

Bibliography

[1] T. C. Redman, "The impact of poor data quality on the typical enterprise," *Communications of the ACM*, vol. 41, no. 2, pp. 79–82, 1998.

[2] I. F. Ilyas, "*Effective Data Cleaning with Continuous Evaluation*," in *IEEE Data Engineering Bulletin*, vol. 39, pp. 38–46, 2016.

[3] P. Howard, "The business case for data quality," in *Bloor Research, White Paper*, 2012.

[4] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "*Duplicate record detection: A survey*," in *IEEE Transactions on Knowledge and Data Engineering*, 2007.

[5] A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti, "Descriptive and prescriptive data cleaning," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 445–456, 2014.

[6] L. Sweeney, "K-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[7] J. Stoyanovich, K. Yang, and H. Jagadish, "Online set selection with fairness and diversity constraints," in *International Conference on Extending Database Technology*, pp. 241–252, 2018.

[8] Y. Huang, F. Chiang, A. Maier, M. Petitclerc, Y. Saillet, D. Spisic, and C. Zuzarte, "Quantifying duplication to improve data quality," in *Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering*, pp. 272–278, 2017.

[9] Y. Huang and F. Chiang, "Refining duplicate detection for improved data quality.," in *International Conference on Theory and Practice of Digital Libraries*, 2017.

[10] Y. Huang and F. Chiang, "Towards a unified framework for data cleaning and data privacy," in *International Conference on Web Information Systems Engineering*, pp. 359–365, Springer, 2015.

[11] Y. Huang, M. Milani, and F. Chiang, "Pacas: Privacy-aware, data cleaning-as-a-service," in *IEEE International Conference on Big Data*, pp. 1023–1030, 2018.

[12] Y. Huang, M. Milani, and F. Chiang, "Privacy-aware data cleaning-as-a-service," vol. 94, 2020.

[13] H. Yao and H. J. Hamilton, "Mining functional dependencies from data," *Data Mining and Knowledge Discovery*, vol. 16, no. 2, pp. 197–219, 2008.

[14] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu, "Towards certain fixes with editing rules and master data," *The VLDB Journal*, vol. 21, no. 2, pp. 213–238, 2012.

[15] W. Fan, X. Jia, J. Li, and S. Ma, "*Reasoning About Record Matching Rules*," in *Proceedings of the VLDB Endowment*, vol. 2, pp. 407–418, 2009.

[16] F. Chiang and R. J. Miller, "A unified model for data and constraint repair," in *IEEE International Conference on Data Engineering*, pp. 446–457, 2011.

[17] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller, "Continuous data cleaning," in *IEEE International Conference on Data Engineering*, pp. 244–255, 2014.

[18] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "Holoclean: Holistic data repairs with probabilistic inference," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1190–1201, 2017.

[19] Z. Yu and X. Chu, "Piclean: A probabilistic and interactive data cleaning system," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 2021–2024, 2019.

[20] S. Kolahi and L. Lakshmanan, "On approximating optimum repairs for functional dependency violations," in *International Conference on Database Theory*, pp. 53–62, 2009.

[21] I. F. Ilyas, X. Chu, *et al.*, "Trends in cleaning relational data: Consistency and deduplication," *Foundations and Trends in Databases*, vol. 5, no. 4, pp. 281–393, 2015.

[22] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu, "Corleone: Hands-off crowdsourcing for entity matching," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 601–612, 2014.

[23] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "Crowder: Crowdsourcing entity resolution," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1483–1494, 2012.

[24] M. Yakout, Elmagarmid, Neville, Ouzzani, and Ilyas, "Guided data repair," *Proceedings of the VLDB Endowment*, vol. 4, no. 5, pp. 279–289, 2011.

[25] X. Chu, Morcos, Ilyas, Ouzzani, Papotti, Tang, and Ye, "KATARA: A data cleaning system powered by knowledge bases and crowdsourcing," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1247–1261, 2015.

[26] P. Samarati, "Protecting respondents identities in microdata release," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 6, pp. 1010–1027, 2001.

[27] I. Bhattacharya and L. Getoor, "*Collective entity resolution in relational data*," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, p. 5, 2007.

[28] W. E. Winkler, "*The state of record linkage and current research problems*," in *Statistical Research Division, US Census Bureau*, 1999.

[29] A. Arasu, M. Götz, and R. Kaushik, "*On active learning of record matching packages*," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 783–794, 2010.

[30] W. E. Winkler, "*Overview of record linkage and current research directions*," in *Bureau of the Census*, 2006.

[31] E. S. Ristad and P. N. Yianilos, "*Learning string-edit distance*," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 522–532, 1998.

[32] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian, "*Metric functional dependencies*," in *IEEE International Conference on Data Engineering*, pp. 1275–1278, 2009.

[33] W. W. Cohen, "*Integration of heterogeneous databases without common domains using queries based on textual similarity*," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol. 27, pp. 201–212, 1998.

[34] O. Benjelloun, H. Garcia-Molina, H. Kawai, T. E. Larson, D. Menestrina, Q. Su, S. Thavisomboon, and J. Widom, "*Generic Entity Resolution in the SERF Project*," in *IEEE Data Engineering Bulletin*, 2006.

[35] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom, "*Swoosh: a generic approach to entity resolution*," in *Proceedings of the VLDB Endowment*, pp. 255–276, 2009.

[36] G. de Assis Costa and J. M. P. de Oliveira, "*A blocking scheme for entity resolution in the semantic web*," in *IEEE 30th international conference on Advanced Information networking and applications*, pp. 1138–1145, 2016.

[37] Q. Wang, M. Cui, and H. Liang, "*Semantic-aware blocking for entity resolution*," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, pp. 166–180, 2016.

[38] S. Benbernou, X. Huang, and M. Ouziri, "*Semantic-based and Entity-Resolution Fusion to Enhance Quality of Big RDF Data*," in *IEEE Transactions on Big Data*, 2017.

[39] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang, "*DeepER–Deep Entity Resolution*," in *arXiv preprint arXiv:1710.00597*, 2017.

[40] S. D. Pradap Konda, P. S. GC, A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, S. Prasad, *et al.*, "*Magellan: Toward Building Entity Matching Management Systems*," in *Proceedings of the VLDB Endowment*, vol. 9, 2016.

[41] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra, "*Deep Learning for Entity Matching: A Design Space Exploration*," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 19–34, 2018.

[42] J. Gardezi, L. Bertossi, and I. Kiringa, "Matching dependencies: semantics and query answering," *Frontiers of Computer Science*, vol. 6, no. 3, pp. 278–292, 2012.

[43] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "*A neural probabilistic language model*," in *Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[44] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "*Efficient estimation of word representations in vector space*," in *International Conference on Learning Representations*, 2013.

[45] D. Holmes and M. C. McCabe, "*Improving precision and recall for soundex retrieval*," in *International Symposium on Information Technology*, pp. 22–26, 2002.

[46] Y. Fu, H. Jiang, N. Xiao, L. Tian, and F. Liu, "Aa-dedupe: An application-aware source deduplication approach for cloud backup services in the personal computing environment," in *IEEE International Conference on Cluster Computing*, pp. 112–120, 2011.

[47] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl, "*Constrained K-means Clustering with Background Knowledge*," in *International Conference on Machine Learning*, pp. 577–584, 2001.

[48] E. Ukkonen, "Approximate string-matching with q-grams and maximal matches," in *Theoretical computer science*, vol. 92, pp. 191–211, Elsevier, 1992.

[49] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "*Crowder: Crowdsourcing entity resolution*," in *Proceedings of the VLDB Endowment*, vol. 5, pp. 1483–1494, 2012.

[50] H. Yao and H. J. Hamilton, "Mining functional dependencies from data," *Data Mining and Knowledge Discovery*, vol. 16, no. 2, pp. 197–219, 2008.

[51] F. Tramer, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. D. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *International Conference on Learning Representations*, 2018.

[52] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas, "Comparative analysis of approximate blocking techniques for entity resolution," *Proceedings of the VLDB Endowment*, vol. 9, no. 9, pp. 684–695, 2016.

[53] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "*xDistributed representations of words and phrases and their compositionality*," in *Conference on Neural Information Processing Systems*, pp. 3111–3119, 2013.

[54] "Google pre-trained word and phrase vectors." `https://code.google.com/archive/p/word2vec/`, 2013. [Online; accessed 20-November-2018].

[55] "Restaurant dataset." `https://www.cs.utexas.edu/users/ml/riddle/data.html`, 2003. [Online; accessed 19-May-2017].

[56] "Benchmark datasets for entity resolution." `http://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution`, 2010. [Online; accessed 20-November-2016].

[57] "stock dataset." `http://www.nasdaq.com/`, 2015. [Online; accessed 20-November-2016].

[58] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[59] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[60] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi, "A cost-based model and effective heuristic for repairing constraints by value modification," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 143–154, 2005.

[61] T. Dasu and M. Loh, "Statistical distortion: Consequences of data cleaning," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1674–1683, 2012.

[62] G. Jagannathan and R. Wright, "Privacy-preserving imputation of missing data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 65, no. 1, pp. 40–56, 2008.

[63] F. Chiang and D. Gairola, "Infoclean: Protecting sensitive information in data cleaning," *Journal of Data and Information Quality*, vol. 9, no. 4, pp. 1–26, 2018.

[64] S. Krishnan, J. Wang, M. J. Franklin, K. Goldberg, and T. Kraska, "PrivateClean: Data cleaning and differential privacy," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 937–951, 2016.

[65] P. Samarati, "Protecting respondents' identities in microdata release," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 6, pp. 1010–1027, 2001.

[66] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu, "Privacy-preserving data publishing: A survey of recent developments," *ACM Computing Surveys*, vol. 42, no. 4, pp. 14:1–14:53, 2010.

[67] C. Ge, X. He, I. F. Ilyas, and A. Machanavajjhala, "Apex: Accuracy-aware differentially private data exploration," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 177–194, 2019.

[68] L. Bertossi and L. Bravo, *Generic and Declarative Approaches to Data Quality Management*, pp. 181–211. Springer Berlin Heidelberg, 2013.

[69] C. Dwork, "Differential privacy," in *International Colloquium on Automata, Languages and Programming*, pp. 1–12, 2006.

[70] S. Deep and P. Koutris, "QIRANA: A framework for scalable query pricing," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 699–713, 2017.

[71] M. Balazinska, B. Howe, and D. Suciu, "Data markets in the cloud: An opportunity for the database community," *Proceedings of the VLDB Endowment*, vol. 4, no. 12, pp. 1482–1485, 2011.

[72] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," *ACM Transactions on Knowledge Discovery from Data*, vol. 1, no. 1, pp. 1–52, 2007.

[73] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 3rd ed., 2011.

[74] S. Lee, S. Huh, and R. D. McNiel, "Automatic generation of concept hierarchies using wordnet," *Expert Systems with Applications*, vol. 35, no. 3, pp. 1132–1144, 2008.

[75] I. K. Fayyad, U.M., "Multi-interval discretization of continuous-valued attributes for classification learning," in *International Joint Conference on Artificial Intelligence*, p. 1022–1027, 1993.

[76] R. Kerber, "Chimerge: Discretization of numeric attributes," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 123–128, 1992.

[77] H. Liu and R. Setiono, "Feature selection via discretization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 4, pp. 642–645, 1997.

[78] AggData-LLC, "AggData locational data." `https://www.aggdata.com/`, 2009.

[79] Infochimps, "Infochimps big data business." `http://www.infochimps.com/`, 2018.

[80] Kaggle, "Kaggle datasets." `https://www.kaggle.com`, 2017.

[81] A. Nash, L. Segoufin, and V. Vianu, "Views and queries: Determinacy and rewriting," *ACM Transactions on Database Systems*, vol. 35, no. 3, pp. 1–41, 2010.

[82] A. Gionis and T. Tassa, "k-anonymization with minimal loss of information," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 2, pp. 206–219, 2009.

[83] A. Roth, "Buying private data at auction: The sensitive surveyor's problem," *ACM SIGecom Exchanges*, vol. 11, no. 1, pp. 1–8, 2012.

[84] F. Chiang and R. J. Miller, "Active repair of data quality rules," in *Proceedings of the International Conference on Information Quality*, pp. 174–188, 2011.

[85] G. Beskales, I. F. Ilyas, and L. Golab, "Sampling the repairs of functional dependency violations under hard constraints," in *Proceedings of the VLDB Endowment*, pp. 197–207, 2010.

[86] L. Golab, I. F. Ilyas, G. Beskales, and A. Galiullin, "On the relative trust between inconsistent data and inaccurate constraints," in *IEEE International Conference on Data Engineering*, pp. 541–552, 2013.

[87] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang, "Nadeef: a commodity data cleaning system," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 541–552, 2013.

[88] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R. J. Miller, and M. Wang, "Linkedct: A linked data space for clinical trials," *preprint arXiv:0908.0567*, 2009.

[89] "Bioportal medical ontology." `https://bioportal.bioontology.org/ontologies`, 2018.

[90] "Disease ontology." `http://disease-ontology.org/`, 2018.

[91] U. of Michigan Library, "Libraries of ontologies." `https://guides.lib.umich.edu/ontology/ontologies`, 2017.

[92] "Census-income database." `https://archive.ics.uci.edu/ml/machine-learning-databases/census-income-mld/census-income.html`, 2017.

[93] "Data and statistics about the U.S.." `https://www.usa.gov/statistics`, 2018.

[94] "Structure of the U.S. education." `https://www.ed.gov/`, 2017.

[95] "New york state food service establishment inspection." `https://www.health.ny.gov`, 2017.

[96] P. Arocena, Glavic, Mecca, Miller, Papotti, and Santoro, "Messing up with bart: error generation for evaluating data-cleaning algorithms," *Proceedings of the VLDB Endowment*, vol. 9, no. 2, pp. 36–47, 2015.

[97] M. Drosou, H. Jagadish, E. Pitoura, and J. Stoyanovich, "Diversity in big data: A review," *Big Data*, vol. 5, no. 2, pp. 73–84, 2017.

[98] G. Aggarwal, Feder, Kenthapadi, Motwani, Panigrahy, Thomas, and Zhu, "Anonymizing tables," in *International Conference on Database Theory*, pp. 246–258, 2005.

[99] A. Meyerson and R. Williams, "On the complexity of optimal k-anonymity," in *Symposium on Principles of Database Systems*, pp. 223–228, 2004.

[100] B. Grau and E. Kostylev, "Logical foundations of privacy-preserving publishing of linked data," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 943–949, 2016.

[101] M. Hay, G. Miklau, D. Jensen, D. Towsley, and C. Li, "Resisting structural reidentification in anonymized social networks," *The VLDB Journal*, vol. 19, no. 6, pp. 797–823, 2010.

[102] S. Barocas and A. Selbst, "Big data's disparate impact," *California Law Review*, vol. 104, no. 671, pp. 671–732, 2016.

[103] L. Sweeney, "Discrimination in online ad delivery," *Communications of the ACM*, vol. 56, 2013.

[104] A. Datta, S. Sen, and Y. Zick, "Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems," in *2016 IEEE Symposium on Security and Privacy*, pp. 598–617, 2016.

[105] R. Nabi and I. Shpitser, "Fair inference on outcomes," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 1931–1940, 2018.

[106] M. Young, L. Rodriguez, E. Keller, F. Sun, B. Sa, J. Whittington, and B. Howe, "Beyond open vs. closed: Balancing individual privacy and public accountability in data sharing," in *ACM Conference on Fairness, Accountability, and Transparency*, p. 191–200, 2019.

[107] V. Bindschaedler, R. Shokri, and C. Gunter, "Plausible deniability for privacy-preserving data synthesis," *Proceedings of the VLDB Endowment*, vol. 10, no. 5, p. 481–492, 2017.

[108] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett, "Differentially private histogram publication," *The VLDB Journal*, vol. 22, no. 6, p. 797–822, 2013.

[109] D. Pujol, R. McKenna, S. Kuppam, M. Hay, A. Machanavajjhala, and G. Miklau, "Fair decision making using privacy-protected data," in *ACM Conference on Fairness, Accountability, and Transparency*, p. 189–199, 2020.

[110] K. Yang and J. Stoyanovich, "Measuring fairness in ranked outputs," in *International Conference on Scientific and Statistical Database Management*, pp. 22:1–22:6, 2017.

[111] R. Caruana, M. Elhawary, N. Nguyen, and C. Smith, "Meta clustering," *International Conference on Data Mining*, pp. 107–118, 2006.

[112] J. M. Phillips, P. Raman, and S. Venkatasubramanian, "Generating a diverse set of high-quality clusterings," in *Proceedings of the International Conference on Discovering, Summarizing and Using Multiple Clusterings*, pp. 80–91, 2011.

[113] J. Li, K. Yi, and Q. Zhang, "Clustering with diversity," in *Automata, Languages and Programming*, pp. 188–200, 2010.

[114] T. Coleman and J. Moré, "Estimation of sparse jacobian matrices and graph coloring problems," *SIAM Journal on Numerical Analysis*, vol. 20, no. 1, pp. 187–209, 1983.

[115] "Pantheon dataset." `https://pantheon.world/`, 2014.

[116] "Uci machine learning repository." `https://archive.ics.uci.edu/ml/datasets/`, 2020.

[117] M. Mannino and A. Abouzied, "Is this real? generating synthetic data that looks real," in *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, p. 549–561, 2019.

[118] "Diva: Extended evaluation details." `https://diva1234567.github.io/DIVA/`, 2020.

[119] J. Byun, A. Kamra, E. Bertino, and N. Li, "Efficient k-anonymization using clustering techniques," in *Database Systems for Advanced Applications*, pp. 188–200, 2007.

[120] R. Bayardo and R. Agrawal, "Data privacy through optimal k-anonymization," in *IEEE International Conference on Data Engineering*, p. 217–228, 2005.

[121] J. Stoyanovich, S. Abiteboul, and G. Miklau, "Data, responsibly: Fairness, neutrality and transparency in data analysis," in *International Conference on Extending Database Technology*, 2016.