Preserving Privacy in High-Dimensional Data Publishing

Khalil Al-Hussaeni

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

Concordia University

Montréal, Québec, Canada

April 2017

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Khalil Al-Hussaeni**

Entitled: **Preserving Privacy in High-Dimensional Data Publishing**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair

Carson K.-S. Leung _____ External Examiner

Todd Eavis _____ Examiner

Lingyu Wang _____ Examiner

Mohammad Mannan _____ Examiner

Benjamin C. M. Fung _____ Supervisor

Rachida Dssouli _____ Supervisor

Approved by: _____

Chair of Department or Graduate Program Director

_____ 20 _____ _____

Amir Asif, Ph.D., Dean

Faculty of Engineering and Computer Science

Abstract

Preserving Privacy in High-Dimensional Data Publishing

Khalil Al-Hussaeni, Ph.D.
Concordia University, 2017

We are witnessing a continuous expansion of information technology that never ceases to impress us with its computational power, storage capacity, and agile mobility. Such technology is becoming more pervasive by the day and has enhanced various aspects of our daily lives. GPS-equipped devices, smart card automated fare collection systems, and sensory technology are but a few examples of advanced, yet affordable, data-generating technologies that are an integral part of modern society. To enhance user experience or provide better services, service providers rely on collecting person-specific information from users. Thus, the collected data is studied and analyzed in order to extract useful information. It is a common practice for the collected data to be shared with a third-party, e.g., a data mining firm, for data analysis. However, the shared data must not leak sensitive information about the individuals to whom the data belongs or reveal their identity. In other words, individuals' privacy must be protected in the published data. *Privacy-preserving data publishing* is a research area that studies anonymizing person-specific data without compromising its utility for future data analysis. This thesis studies and proposes anonymization solutions for three types of *high-dimensional* data: *trajectory streams*, *static trajectories*, and *relational data*. We demonstrate through theoretical and experimental analysis that our proposed solutions, for the most part, outperform state-of-the-art methods in terms of utility, efficiency, and scalability.

Dedication

To my family, with love.

Acknowledgments

This thesis is dedicated to my beloved family for all their support and unconditional love. Mom, Dad, your emotional support throughout this journey of mine has been truly unparalleled. You have shown me limitless love, and it has gotten me where I am. To my dearest brothers, who stood by my side and lifted up my spirits when things seemed far-fetched, I love you, and I will always be your caring brother throughout your own journeys.

I would like to extend my sincere gratitude to my supervisor, Dr. Benjamin C. M. Fung, who has been a constant source of inspiration to me. I learned a lot from him, not just from his vast technical knowledge and stimulating discussions, but also from his management skills, especially towards his students. Dr. Fung, I thank you for your guidance, patience, kind words, constant respect, and never giving up on me. Moreover, my utmost respect goes to my co-supervisor, Dr. Rachida Dssouli, who never hesitated to extend her kindness to me and to support me in every way she could.

Last, but not least, I would like to express my gratitude to Dr. Ibrahim Kaka for his unwavering benevolence and for being the inspirational epitome of selflessness.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent years have witnessed a tremendous growth in data collection thanks to the exponential development of information technology that not only facilitates our daily life, but also generates extensive amounts of person-specific data. Data of different types are generated on a daily basis, such as GPS data [92] [88], RFID data [42] [55] [71] [124], moving objects or trajectory data [17] [1] [139] [51], health-care data [82] [113], search queries [14], and customers' purchases [18] [138]. There has been a compelling demand in various sectors for collecting such data, whether by government agencies, transportation authorities, medical centers, online service providers, or retail corporations. This demand stems from the need to analyze the collected data and extract useful information in order to construct a knowledge base that helps in decision-making operations. Usually, the collected data is published to a third party, e.g., a research center, in order to conduct the desired analysis.

We call the entity that collects, holds, and publishes the data a *data holder* or a *data publisher*, the individuals from whom the data is collected *record owners* (each record in a tabular database belongs to a unique individual), the entity that receives the published data a *data recipient*, and data recipients who attempt to perform a privacy attack *adversaries* or *attackers*. Figure 1.1 illustrates the primary

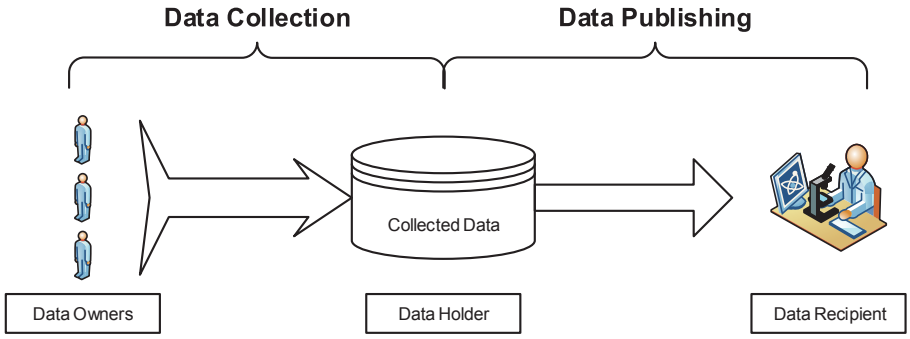


Figure 1.1: Data collection and publishing

participants in a data collection and publishing scenario. For example, a search engine company (data holder) may want to publish queries submitted by users (data owners) to the public (data recipients) in order to improve the company's recommendation system. The terms *data publishing* and *data release* refer to the same process and will be used interchangeably in this thesis.

Publishing collected data can serve a variety of purposes and can be mutually beneficial to both the data holder and data recipient. For example, experts mining released road traffic data can extract interesting patterns, which in turn helps in improving the city's road network and, thus, reducing traffic jams. Another interesting example is Netflix, the giant movie rental service, which released a dataset containing movie ratings that belong to nearly 500,000 subscribers and offered an enticing prize for anyone who could improve their recommendation service [57]. In some cases, publishing data is mandated by law. For example, there is a regulation in California that requires licensed hospitals to publish records of discharged patients [23]. Generally speaking, whether the published data is shared with a private entity or is available for public access, it is significantly beneficial for researchers and practitioners to test a proposed solution on real-life data, as opposed to computer-generated, synthetic data.

In most cases, the collected data belongs to real-life individuals, and publishing their data may lay their privacy on the line. The published data must not allow

individuals' identities to be exposed, nor their private information to be linked to them, whether accidentally or deliberately. A *privacy attack* is, thus, the ability to perform such *linkage*. Therefore, before releasing the data, data holders tend to *anonymize* their collected data by removing explicit identifiers such as *Name*, *Date of Birth*, and *SSN*. The objective is to prevent linking sensitive information to an individual. Though seemingly benign, such data anonymization has been proven inadequate in protecting individuals' privacy.

In the following section we will present some classical examples that show that simply removing explicit identifiers fails to protect the privacy of individuals whose data is being published.

1.1 Motivation

Among the most common examples of privacy attacks is the incident that took place in the state of Massachusetts [114]. A government agency, Group Insurance Commission (GIC), bought health insurance for state employees. For the sake of offering real-life data for researchers, GIC then decided to publish a dataset composed of hospital visits of every state employee. For privacy protection, GIC removed all names, addresses, and other explicit identifiers from the published dataset. Sweeney [114], a professor of computer science, purchased an unanonymous copy of the state's voter list for $20. From both the published dataset and voter list, Sweeney's astonishing finding was that 87% of the U.S. population can be uniquely identified using the combination of date of birth, gender, and ZIP code. Based on this study, Sweeney was able to identify from the published dataset the medical record of Governor William Weld. Sweeney's finding was based on the 1990 census data. A decade after, Golle [54] recalculated the statistics based on the 2000 census data. Using the same combination of *quasi-identifier attributes* (*QID*), i.e., date of birth, gender, and ZIP code, Golle was able to uniquely identify 63% of the U.S. population.

In 2006, Netflix, the famous online movie rental service, offered a prize of $\$1,000,000$ to whoever could improve their movie recommendation system by 10% [57]. In order to facilitate the job for researchers, Netflix released a dataset containing movie ratings that belong to nearly 500,000 subscribers, along with the movie titles and the rating of each movie [101]. In an attempt to protect users' privacy, all explicit identifiers were removed from the published dataset and replaced by randomly-assigned IDs. However, the work in [101] suggests that knowing the dates ($\pm 2 weeks$) of six movie ratings, 99% of the people in the published dataset are identifiable. Furthermore, knowing only two movies with their rating dates ($\pm 3 days$), 68% of the subscribers are identifiable from the dataset. The authors in [101] performed their attack by using a publicly-available, unanonymous, external dataset of movie reviews from the Internet Movie Database (IMDb) website.

The above two examples demonstrate a privacy attack facilitated by the possibility of collecting *background knowledge* from external data sources that contain shared information about the same group of individuals in the published dataset. The next example demonstrates the possibility of uniquely identifying an individual in seemingly anonynmous data.

America Online (AOL) released a dataset containing 20 million search queries of 650,000 users. The purpose behind this data release was "to embrace the vision of an open research community". AOL anonymized the data before publishing by removing AOL usernames, IP addresses, and other identifiers. A random number was given to each user and his/her searches. In the published dataset, one user was unlucky enough to submit a set of queries, some of which include: "landscapers in Lilburn, GA", "several people with the last name Arnold", and "homes sold in shadow lake subdivision gwinnett county georgia". Those queries turned out to be so unique among all other submitted queries that two New York Times reporters quickly identified who was looking for "homes sold in shadow lake subdivision gwinnett county georgia"; Ms. Thelma Arnold, a 62-year-old widow who lives in Lilburn,

Georgia [14]. In an interview, Ms. Thelma Arnold then acknowledged the searched queries. This incident led to the retrieval of the published dataset and made data holders reluctant to release their data, at least for the foreseeable future.

1.2 Privacy-Preserving Data Publishing

Privacy-preserving data publishing (*PPDP*) [44] is the study of applying adequate anonymization measures to the data to be published, taking into consideration *two factors*: protecting individuals' privacy in the published data and maintaining high data utility for accurate analysis of the anonymous data. Privacy protection is achieved by transforming the *raw data* to another anonymous version that adheres to some privacy requirements dictated by the applied privacy model; i.e., this process is referred to as *anonymizing* the data. Preserving data utility concerns the cost, in terms of data quality degradation, ensued due to anonymizing the raw data.

The importance of PPDP stems from the need to share real-life data with researchers, data analysts, or the general public, while safeguarding individuals' privacy in the published data. Moreover, we do not want data holders to hesitate when publishing their data because such real-life data is crucial for obtaining meaningful and accurate data mining results, e.g., when performing classification analysis [125]. Therefore, PPDP ensures that useful, yet privacy-preserving, data can be shared and benefited from in various sectors.

In this thesis, *raw data* refers to the original collected data that includes person-specific and sensitive pieces of information about some individuals and is kept in the data holder's possession. We assume that the data holder wishes to first anonymize, then publish the collected data. The published data, which has been anonymized before publishing, is described as being *anonymous* or *sanitized*. We use these two terms interchangeably in this thesis.

1.3 Contributions

Due to the pervasiveness of information technology that constantly generates person-specific data, and due to the compelling demand for sharing such data in various sectors, we present a comprehensive study that spans different PPDP scenarios. Particularly, this thesis explores publishing three types of *high-dimensional* data: *trajectory streams*, *static trajectory data*, and *relational data*. The *curse of high dimensionality* [3] describes a certain type of data containing a sufficiently large number of QID attributes. Such data is characterized by being sparse in the high-dimensional space, and anonymizing sparse data without compromising its utility is a challenging problem. This is due to the fact that higher dimensionality results in more unique data points in the high-dimensional space. Consequently, such uniqueness has to be masked by completely removing or altering original data points in order to provide sufficient privacy protection, rendering the anonymous data poor in utility. The following is a summary of the primary contributions of this thesis.

1.3.1 Anonymizing Trajectory Streams

Recent advancement in mobile computing and sensory technology has facilitated the possibility of continuously updating, monitoring, and detecting the latest location and status of moving individuals. Spatio-temporal data generated and collected on the fly is described as *trajectory streams*. This work is motivated by the concern that publishing individuals' trajectories on the fly may jeopardize their privacy.

In this thesis, we illustrate and formalize two types of privacy attacks against moving individuals. We make three observations about trajectory streams and devise a novel algorithm, called *Incremental Trajectory Stream Anonymizer* (*ITSA*), for incrementally anonymizing a sequence of sliding windows, which are dynamically updated with joining and leaving individuals. The update process is done by employing an efficient data structure to accommodate large-volume streaming data.

We conduct extensive experiments on simulated and real-life datasets. When compared with closely-related anonymization methods, experimental results demonstrate that our method significantly lowers runtime and efficiently scales when handling massive trajectory datasets. Moreover, experiments on real-life web logs suggest that our method can seamlessly extend to high-dimensional transaction data. To the best of our knowledge, this thesis presents the first work to anonymize high-dimensional trajectory streams.

1.3.2 Anonymizing Static Trajectories

In recent years, the collection of spatio-temporal data that captures human movements has increased tremendously due to advancements in hardware and software systems capable of collecting user-specific data. The bulk of the data collected by these systems has numerous applications, including general data analysis. Therefore, publishing such data is greatly beneficial for data recipients. However, in its raw form, the collected data contains sensitive information pertaining to the individuals from whom it was collected and must be anonymized before publication.

In this thesis, we study the problem of privacy-preserving trajectory publishing and propose a solution under the rigorous differential privacy model [34]. Unlike sequential data, which describes sequentiality between data items, handling spatio-temporal data is a challenging task due to the fact that introducing a temporal dimension results in extreme sparseness. Our proposed solution introduces an efficient algorithm, called *SafePath*, that models trajectories as a noisy prefix tree and publishes ϵ-differentially-private trajectories while minimizing the impact on data utility. Experimental evaluation on real-life transit data from the *Société de transport de Montréal* (STM) suggests that *SafePath* significantly improves efficiency and scalability with respect to large and sparse datasets, while achieving comparable results to existing solutions in terms of the utility of the sanitized data.

1.3.3 Anonymizing Relational Data

Various organizations collect data about individuals for various reasons, such as service improvement. In order to mine the collected data for useful information, it has become a common practice to share the collected data with data analysts, research institutes, or simply the general public. The quality of published data significantly affects the accuracy of the data analysis, and thus affects decision making at the corporate level.

In this thesis, we propose *DiffMulti*, a workload-aware algorithm that employs *multidimensional generalization* under differential privacy. We devise an efficient implementation to the proposed algorithm and use a real-life dataset for experimental analysis. We show that the computational cost of *DiffMulti* is bounded by the number of records rather than by the number of attributes of an input dataset, making our proposed method suitable for high-dimensional data. We evaluate the performance of our method in terms of data utility, efficiency, and scalability. Experimental comparisons with related anonymization methods from the literature suggest that *DiffMulti* is capable of improving data utility, in some cases, by orders of magnitude, while performing comparably, at worst, in terms of efficiency and scalability.

1.4 Organization of the Thesis

We divide the presentation of the problems in this thesis according to the applied privacy model. We first present the problem of anonymizing trajectory streams under a *syntactic* privacy model. We then proceed to present the second problem, anonymizing static trajectories, and the third problem, anonymizing relational data, under a *semantic* privacy model. The rest of the thesis is organized as follows:

- Chapter 2 demonstrates through examples some privacy attacks, introduces

some popular privacy models to counter such attacks, and examines widely-used anonymization techniques.

- Chapter 3 explores prominent works that have been proposed for protecting individuals' privacy in the domain of privacy-preserving data publishing [44]. Particularly, we present in-depth literature review of existing anonymization methods for four types of data: data streams, trajectory data, transaction data, and relational data.

- Chapter 4 studies the problem of publishing data streams, particularly, trajectory data streams. We identify and formalize three properties in a trajectory stream, and we integrate these properties in building an efficient algorithm that incrementally anonymizes the transient data over a sequence of sliding windows. To the best of our knowledge, this chapter presents the first work to anonymize high-dimensional trajectory streams. The results of this chapter have been published in [6].

- Chapter 5 studies the problem of publishing high-dimensional and sparse static trajectory data. We model trajectories in a noisy prefix tree structure, which provides concise representation of the underlying sparse data. The results of this chapter are currently under review in [7].

- Chapter 6 proposes a differentially-private method for anonymizing relational data. Our method leverages the promising anonymization technique of multi-dimensional generalization, which enhances the utility of the anonymous data without compromising the computational cost. The work presented in this chapter is currently under review in [8].

- Chapter 7 concludes this thesis.

Chapter 2

Background

In this chapter, we present some fundamental privacy concepts. First, we demonstrate through examples some privacy attacks that rely on pre-existing knowledge about the targeted individuals in the published data. Then, we introduce existing privacy models that were proposed to thwart those attacks or mitigate their impact on individuals' privacy. Finally, we examine some widely used techniques that enforce those models and transform the input data from its raw form to its anonymous form under some given privacy requirements.

2.1 Privacy Attacks

When publishing data tables that contain information about a certain group of individuals, it is equally important to protect the privacy of those individuals whose data is being shared. The term *privacy* in this context is twofold: (1) the *identity* of any individual in the published data should not be revealed, and (2) any piece of information deemed *sensitive* should not be associated with its pertinent individual.

We present two types of privacy threats that jeopardize the privacy of individuals whose data is being published. If the anonymization model does not account for how much information can be gathered from external data sources about the

individuals in the published data, then the published data is susceptible to *record linkage* and *attribute linkage* attacks. These attacks primarily target relational data tables, where a row represents a unique individual and a column represents an attribute that describes a certain type of information about the individuals in the table. However, linkage attacks can be extended to other types of data, such as transaction data [118] [119] [138] [50] and trajectory data [103] [1] [99].

2.1.1 Record Linkage

Record linkage refers to the type of privacy attacks that aim at uniquely identifying a group of individuals in a published data table with the aid of previously-collected information about those individuals. Sweeney [114] presented an attack scenario in which a malicious data recipient gathers information about a group of individuals in the published data table using some external data source. Such externally-acquired knowledge, called the attacker's *background knowledge*, is then used to single out (or narrow down) a target victim's record in the published data table. A combination of certain attributes, known as *quasi-identifiers* (*QID*), in the published table can potentially lead to the re-identification of a certain group of individuals. The QID attributes are common attributes between the published data and the external data.

Example 2.1.1. Suppose a hospital is publishing its patients' records to a research center for data analysis. Table 2.1(a) shows the published raw table after removing explicit identifiers. Let attributes YoB (Year of Birth), $Gender$, and Job be the QID attributes in this scenario. Suppose an attacker learned, through an unanonymous external data table, that his neighbor, Bob, has the following values over the QID attributes $\langle 1955, Male, Architect \rangle$. Such a combination uniquely identifies the patient in record 1. Consequently, the attacker now knows that record 1 in Table 2.1(a) belongs to Bob and that he has been diagnosed with HIV. ■

Table 2.1: Illustrating record and attribute linkage attacks on raw data

(a) Raw patients table

	YoB	Gender	Job	Disease
1	1955	Male	Architect	HIV
2	1953	Male	Lawyer	Flu
3	1955	Male	Writer	HIV
4	1951	Male	Painter	Flu
5	1961	Female	Painter	Flu
6	1965	Female	Writer	HIV
7	1965	Female	Writer	HIV

(b) 2-anonymous patients table

	YoB	Gender	Job	Disease
1	[1950-1960)	Male	Professional	HIV
2	[1950-1960)	Male	Professional	Flu
3	[1950-1960)	Male	Artist	HIV
4	[1950-1960)	Male	Artist	Flu
5	[1960-1970)	Female	Artist	Flu
6	[1960-1970)	Female	Artist	HIV
7	[1960-1970)	Female	Artist	HIV

2.1.2 Attribute Linkage

Attribute linkage attacks aim at associating sensitive pieces of information with their pertinent individuals. An attribute is deemed sensitive by the data publisher, depending on the severity of the information described by that attribute. Examples of sensitive attributes include salary, disease, and welfare. We note that not necessarily all the domain values of a sensitive attribute are sensitive; it is possible to specify only some domain values as being sensitive. Values deemed sensitive are not to be associated with their corresponding individuals. For example, in Table 2.1(a), attribute $Disease = \{HIV, Flu\}$ is considered to be sensitive. However, in some examples we will consider Flu to be non-sensitive. Even if a patient's record is not unique in the published table, it is possible for an attacker to deduce a target victim's disease. We call the degree of certainty of an attacker deducing a victim's sensitive value the *inference confidence*.

Example 2.1.2. Consider the patients' records in Table 2.1(a). Suppose that an attacker knows that the target patient's record, Alice, contains $\langle 1965, Female, Writer \rangle$. Despite the existence of two records with the same set of values over the QID attributes, both of these records have HIV on the sensitive attribute $Disease$. Therefore, the attacker is able to infer with 100% certainty that Alice has HIV, i.e., the attacker's inference confidence is 100%. ■

Table 2.2: Illustrating minimality attack on anonymous data

(a) Raw table

	Gender	Job	Disease
1	Male	Lawyer	Flu
2	Male	Lawyer	Flu
3	Male	Lawyer	Flu
4	Female	Writer	HIV
5	Female	Writer	HIV

(b) Published anonymous table

	Gender	Job	Disease
1	M-F	Working	Flu
2	M-F	Working	Flu
3	M-F	Working	Flu
4	M-F	Working	HIV
5	M-F	Working	HIV

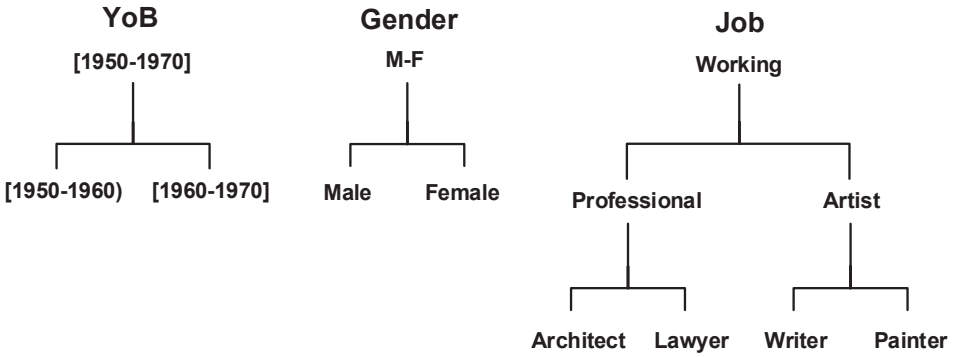


Figure 2.1: Taxonomy trees for patients data

2.1.3 Minimality Attack

As demonstrated in the above two sections, if not enough care is taken to protect individuals' privacy, then their published data table is left vulnerable to linkage attacks. As a result, the raw (original) data table is to be transformed to an anonymous version that satisfies some privacy requirements and, thus, prevents linkage attacks. This process is called anonymization. Some privacy models and anonymization techniques will be discussed in Chapters 2.2 and 2.3, respectively.

Wong et al. [126] discovered that even after anonymizing a data table, an attacker can still associate sensitive values with their corresponding individual. We illustrate Wong et al.'s attack in the following example.

Example 2.1.3. Let us consider a simple version of patients data in Table 2.2(a), which has *Gender* and *Job* as the QID attributes. Let HIV be the only sensitive

value on *Disease*. Under attribute linkage attack, Table 2.2(a) in its raw version reveals with 100% confidence that if the target victim is a $\langle Female, Writer \rangle$ then she has HIV. Suppose we want to lower the inference confidence to 70%; consequently, Table 2.2(a) is anonymized to Table 2.2(b) according to the set of taxonomy trees in Figure 2.1. Table 2.2(b) contains the topmost general values on all the QID attributes, and this results in having only one QID group $\langle M - F, Working \rangle$ in which an attacker can infer that a patient has HIV with 40% confidence at most.

One might think that, since Table 2.2(b) contains very general information and a relatively small inference confidence, the table is well-protected against attribute linkage attacks. Wong et al. [126] argued that this might not necessarily be true. Their reasoning justifiably extends the attacker's knowledge not only to external data tables containing the same QID attributes as the anonymous table, but also to the anonymization algorithm, the imposed privacy requirements, and the set of taxonomy trees. Thus, in this example the attacker possesses the following pieces of information:

1. An external data table similar to Table 2.2(a) minus the *Disease* attribute (for simplicity, assume the same record ordering).

2. The requirement of limiting the inference confidence of HIV to 70%, since it is public knowledge.

3. The published anonymous table, Table 2.2(b).

4. The set of taxonomy trees, as depicted in Figure 2.1.

Given these four pieces of information, the attacker notices that the published table, Table 2.2(b), has been anonymized. Thus, the attacker easily deduces that either the group containing records 1-3 violates the imposed privacy requirement or the group containing records 4-5 does. If the first group had two patients with HIV, then the inference confidence from that group would be $2/3 = 66.67\%$, which

satisfies the privacy requirement; hence, no anonymization is *necessary*. However, if the second group (records 4-5) contains the two HIV values, then the inference confidence is 100%. The attacker deduces that the anonymization took place because records 4-5 must have patients with HIV; otherwise, no anonymization is *necessary*. ■

The above example describes a typical scenario of the *minimality attack* [126]. This type of attacks exploits the fact that existing anonymization algorithms follow the minimality principle when transforming a data table [75]. Under the minimality principle, a raw data table is transformed to its least anonymous version. This is because anonymizing a data table imposes information loss on the original data values; thus, to prevent further distortions, no extra anonymization is performed on an already anonymous table that satisfies some given privacy requirements.

2.2 Privacy Models

2.2.1 k-Anonymity

To protect against record linkage attacks, Samarati and Sweeney [110] [115] proposed k-anonymity. This privacy model states that every record in the published data table must be indistinguishable from at least $k-1$ other records over the QID attributes. Consequently, the maximum probability of a successful record linkage attack is bound to $1/k$. A *QID group* is the set of records that share the same values over the QID attributes. Thus, k-anonymity hides a unique record within a QID group of size $\geq k$. The strength of k-anonymity relies on k and the QID attributes: higher values of k minimize the probability of a successful attack, and more QID attributes provide better protection against a wider range of background knowledge.

Example 2.2.1. Table 2.1(b) is an anonymous version of Table 2.1(a) for $k = 2$. In this case, we say that Table 2.1(b) is 2-anonymous because every record in its

entirety (over the QID attributes) appears at least 2 times in the table. We note that Table 2.1(b) was achieved by generalizing some attribute values in Table 2.1(a) according to the set of taxonomy trees in Figure 2.1. ■

2.2.2 ℓ-Diversity

To thwart attribute linkage attacks, Machanavajjhala et al. [89] proposed the notion of ℓ-diversity. In an ℓ-diverse table, every QID group is required to have at least ℓ "well-represented" records with respect to the sensitive values. ℓ-Diversity is a privacy concept that can be realized in more than one way. The simplest interpretation of "well-represented" is for every QID group in a table to have a minimum of ℓ sensitive values. Let us look at Table 2.1(b), which is a 2-anonymous version of Table 2.1(a). Table 2.1(b) contains 3 QID groups. Let us assume that all domain values of the sensitive attribute *Disease* are sensitive and are not to be associated with their pertinent patients. Since every QID group in Table 2.1(b) contains two different sensitive values, Table 2.1(b) is said to be 2-diverse.

Another interpretation to the ℓ-diversity model is *entropy ℓ-diversity*, which measures how evenly distributed sensitive values are in a table. A more evenly distributed sensitive attribute among table records implies more uncertainty in inferring sensitive values about individuals. In contrast, a less evenly distributed sensitive attribute implies that there are more occurring values in the table than others, thus giving an attacker higher chance in inferring those frequent values. If the table is divided into QID groups, then entropy ℓ-diversity is applied to the sensitive values within each QID group. Machanavajjhala et al. [89] stipulated that for a table to be entropy ℓ-diverse, $log(\ell)$ should be at most equal to the sum of entropies of every sensitive value *sen_val* in each QID group. That is, given ℓ, a table satisfies entropy

ℓ-diversity if the following inequality holds for every QID group in the table:

$$-\sum_{sen\_val \in SA} P(QID, sen\_val) log(P(QID, sen\_val)) \geq log(\ell), \tag{2.1}$$

where SA is the domain of the sensitive attribute and $P(QID, sen\_val)$ is the fraction of records containing the sensitive value sen_val in the QID group. We illustrate entropy ℓ-diversity in the following example.

Example 2.2.2. Let us find the proper value of ℓ that makes Table 2.1(b) satisfy entropy ℓ-diversity. The table contains 3 QID groups; therefore, we need to find the sum of entropies of every value on the sensitive attribute $Disease$ in each QID group. The first group $\langle [1950-1960), Male, Professional \rangle$ gives an entropy sum $= 2 \times (-\frac{1}{2} log(\frac{1}{2})) = log(2)$, the second group $\langle [1950-1960), Male, Artist \rangle$ gives an entropy sum $= 2 \times (-\frac{1}{2} log(\frac{1}{2})) = log(2)$, and the last group $\langle [1960-1970), Female, Artist \rangle$ gives an entropy sum $= -\frac{1}{3} log(\frac{1}{3}) - \frac{2}{3} log(\frac{2}{3}) = log(1.89)$. The last group gives the lowest entropy sum, implying that sensitive values are less evenly distributed than in the other two groups. Hence, Table 2.1(b) is entropy ℓ-diverse only if $\ell \leq 1.89$. ■

A drawback in entropy ℓ-diversity is that it does not provide an intuitive measure for the diversity of the sensitive values. Example 2.2.2 shows that Table 2.1(b) satisfies entropy ℓ-diversity if $\ell \leq 1.89$. However, saying that Table 2.1(b) is entropy 1.89-diverse does not convey the fact that there exists a QID group with 2 out of 3 individuals having HIV. In other words, entropy 1.89-diverse does not point out that an attribute linkage attack has a success probability of 66.67%, a considerably high inference confidence for a disease as sensitive as HIV.

2.2.3 Confidence Bounding

Wang et al. [121] proposed *confidence bounding*, a more flexible and intuitive privacy model than ℓ-diversity. The objective is to ensure that within any QID group

the confidence of inferring a sensitive value $sen\_val \in SA$ does not exceed h, a pre-defined confidence threshold determined by the data holder. The rule for every sensitive value is given by a *privacy template* of the form $\langle QIDgroup \rightarrow sen\_val, h \rangle$, where $QIDgroup \rightarrow sen\_val$ represents the fraction of records containing the sensitive value sen_val in the provided $QIDgroup$. Let us revisit Table 2.1(b). Let $\langle QIDgroup \rightarrow HIV, 50\% \rangle$ be a privacy template defined for HIV. This privacy template specifies that a record linkage attack on HIV should have a maximum success probability of 50%. In both the first and second QID groups, the inference confidence of $sen\_val = HIV$ is $1/2 = 50\%$. However, the third QID group violates this privacy template because the inference confidence of HIV is $2/3 = 66\% > 50\%$. Therefore, Table 2.1(b) does not satisfy the enforced privacy requirement for HIV.

2.2.4 m-Confidentiality

To thwart minimality attacks against published ℓ-diverse tables, Wong et al. [126] proposed *m-confidentiality* to limit the attacker's inference confidence when associating a set of sensitive values with certain individuals in the table. To achieve an m-confidential table, Wong et al. [126] devised an algorithm, called *MASK*, that first transforms the raw data table to a k-anonymous version, where k is a user-defined parameter. Some of the QID groups in the resultant k-anonymous table satisfy the imposed ℓ-diversity requirement, while other QID groups fail to do so. *MASK* then changes the distribution of the sensitive values in the non-ℓ-diverse QID groups based on the distributions of the sensitive values in the ℓ-diverse QID groups in the table. This step results in all QID groups being ℓ-diverse, and thus the entire table satisfies ℓ-diversity and is m-confidential.

Example 2.2.3. Continuing from Example 2.1.3, we showed that Table 2.2(b) is susceptible to minimality attack. Under m-confidentiality, Table 2.2(a) does not need to be anonymized because it already satisfies 2-anonymity. Moreover, the QID group that contains records 4-5 has an inference confidence of 100%; therefore, one

HIV value will be replaced by a non-sensitive value in the distorted record. The resultant table is now ready to be published. ■

m-Confidentiality achieves anonymity with respect to ℓ-diversity by distorting sensitive values. This practice results in less truthful, or even wrong at worst, statistical results, which in turn adversely affect data analysis.

Cormode et al. [32] studied the minimality attack case and provided an in-depth analysis of the algorithms that had been suggested by Wang et al. [121] to be susceptible to such type of attacks. Cormode et al. [32] defined a set of criteria that, if exists in an anonymization algorithm, makes it susceptible to minimality attacks. Moreover, they showed that the attacker's inference confidence does not grow significantly from anonymous data tables that are not m-confidential. Cormode et al.'s study concluded that many anonymization algorithms are not, in fact, vulnerable to minimality attacks, and minor modifications can be made to vulnerable algorithms to mitigate the impact of such attacks.

2.2.5 Differential Privacy

All the above privacy models rely on estimating the attacker's background knowledge in order to thwart linkage attacks. Such type of privacy models is referred to as syntactic-based models because the anonymous data must adhere to some syntactic conditions. To eliminate the effect of any attacker's *power* and avoid devising privacy models with protective strength tied to such power, Dwork proposed *differential privacy* [34]. Differential privacy is a probabilistic privacy model that works independently of any attacker's background knowledge and computational power. In this spirit, differential privacy bounds the probability of obtaining the same answer from two different input datasets, D and D', that differ by *only* one record. Any privacy leak on the differentially-private dataset, denoted by $\hat{D}$, will not be conclusive, as $\hat{D}$ could have been obtained from sanitizing either D or D'. This

gives incentive for individuals to participate in the dataset because a differentially-private mechanism is impartial to the data contained in the input raw dataset. This guarantee, however, is only given under the assumption that no dependencies exist between data records, which is the assumption we adopt in this thesis. For methods that incorporate the dependency assumption under differential privacy, we refer the reader to [87] [28]. We provide the following formal definition:

Definition 2.1 (ϵ-*Differential privacy*)**.** A randomized algorithm Ag gives ϵ-differential privacy if for any neighboring datasets D and D' differing by at most one record, and for any possible output dataset $\hat{D}$,

$$\Pr[Ag(D) = \hat{D}] \leq e^{\epsilon} \times \Pr[Ag(D') = \hat{D}], \tag{2.2}$$

where the probability is taken over the randomness of the Ag. ■

Differential privacy introduces the concept of a *privacy budget*, ϵ. The privacy budget calibrates the utility of the anonymous data. Typically ranging $0 < \epsilon \leq 1$, lower values of ϵ cause more noise to be added to the true answer, and vice versa. The literature has defined two settings whereby ϵ can be utilized: *interactive* and *non-interactive*. In the interactive setting [37] [41], the raw data is kept in the data holder's possession, and a data miner/requester issues a set of queries to which the data holder provides differentially-private answers. Each query would consume a portion of ϵ. Once the entire budget has been consumed, the data holder can no longer receive more queries, and the database has to shut down completely. Whereas in the non-interactive setting [13] [130] [60], the data holder utilizes the entire privacy budget to anonymize the entire raw dataset, and the ϵ-differentially-private version is then published without restriction or limitation on data usage. In many real-life data sharing scenarios, publishing the data is far more convenient due to the flexibility given to data recipients in terms of analysis power. Hence, in this thesis we focus on the non-interactive setting.

Differential privacy has two **composition** properties: *sequential composition* and *parallel composition*. Sequential composition stipulates that if a sequence of differentially-private computations takes place on *the same* set of data, then the entire sequence guarantees the collective privacy guarantee of every computation in the sequence. Whereas, parallel composition applies to situations where a sequence of differentially-private computations is performed on *disjoint* sets of data. In this case, the entire sequence gives the worst privacy guarantee, i.e., the highest privacy budget among the parallel computations.

Lemma 2.1 (*Sequential composition* [93])**.** Let each computation Ag_i provide ϵ_i-differential privacy. A sequence of $Ag_i(D)$ over the dataset D provides $(\sum_i \epsilon_i)$-differential privacy. ■

Lemma 2.2 (*Parallel composition* [93])**.** Let each computation Ag_i provide ϵ_i-differential privacy. A sequence of $Ag_i(D_i)$ over a set of disjoint datasets D_i provides $(max\{\epsilon_i\})$-differential privacy. ■

2.3 Anonymization Techniques

Given a raw data table that contains various types of information about some group of individuals, a data holder wishes to publish the table with the goal of benefiting researchers and data analysts from the collected data. At the same time, the privacy of those individuals should not be compromised. We discussed in Chapter 2.2 some of the most prominent models that shield a published table, if it adheres to a given privacy model, against privacy attacks. The act of transforming a data table to another version that satisfies the requirements of a given (set of) privacy model(s) is called anonymization. The literature has proposed several anonymization techniques; in this section we focus on three primary techniques: *suppression*, *generalization*, and *perturbation*.

Table 2.3: 2-anonymous version of Table 2.1(a) using suppression

	YoB	Gender	Job	Disease
1	1955	Male	*	HIV
3	1955	Male	*	HIV
4	*	*	Painter	Flu
5	*	*	Painter	Flu
6	1965	Female	Writer	HIV
7	1965	Female	Writer	HIV

2.3.1 Suppression

Suppression is a technique whereby some data values are *removed* from the table to be published. When a data value is removed, it is either dropped from the record or replaced by a character/symbol, such as "*". For example, applying 2-anonymity to Table 2.1(a) using suppression yields Table 2.3. Notice that record 2 has been entirely suppressed because it fails to satisfy 2-anonymity. If we are to follow a strict interpretation of k-anonymity, i.e., every record has to appear at least k times over the *entire* set of QID attributes, then we are right to suppress the entire record 2 from Table 2.3. Although, if we are to follow a more relaxed interpretation, then we can keep record 2 in the form $\langle$ *, $Male$,* $\rangle$, where "*" indicates an empty cell.

Suppression is applied in different ways, depending on the desired balance between the utility of the anonymous data and the complexity of the applied algorithm. *Value suppression* [121] removes all the instances of the value to be suppressed from the data table. *Local suppression* [95] [27], also referred to as *cell suppression*, on the other hand, may keep some of the instances of the value to be suppressed from the data table. Intuitively, local suppression incurs less information loss than value suppression; however, the former type comes at the cost of high computational complexity. Lastly, *record suppression* [15] [68] [110] removes entire records, i.e., outliers.

2.3.2 Generalization

Unlike suppression, which removes data values from the data table, generalization replaces violating values (those that do not meet the privacy requirement) by less specific, yet semantically similar, values. Replacing a specific value in an attribute by its more abstract version is done in accordance with a pre-defined generalization hierarchy. For example, Figure 2.1 contains three taxonomy trees; each describes the hierarchy of domain values in an attribute. This set of taxonomy trees is used to generalize the violating data values in Table 2.1(a) to Table 2.1(b) in order to satisfy 2-anonymity.

Several schemes were proposed in the literature to implement generalization. In *global recoding* or *global generalization*, if a value is chosen to be generalized, then all the instances of that value in the data table are generalized. *Local recoding* or *local generalization* generalizes only some instances of the chosen value. *Full-domain generalization* [75] [110] [115] is a global generalization approach whereby if a domain value is chosen to be generalized and is generalized to a certain tree level, then all the other domain values of that attribute are generalized to the same tree level, as well. For example, in Figure 2.1, if $Writer$ is generalized to $Artist$, then (1) all instances of $Writer$ in the data table are generalized to $Artist$, and (2) all instances of $Architect$ and $Lawyer$ are generalized to $Professional$. Although full-domain generalization can achieve optimal solutions (e.g., Incognito [75]), it causes considerable data distortion due to the fact that non-violating data values are generalized along with violating values. In *Subtree generalization* [15], if a data value is chosen to be generalized, then all its siblings in the taxonomy tree will be generalized to the same parent value. For example, in Figure 2.1, if $Writer$ is generalized to $Artist$, then $Painter$ is generalized to artist, and the rest of the domain values remain intact. *Sibling generalization* [75], on the other hand, generalizes only the violating values while their siblings remain intact. Consequently, the data table is less distorted, but a (general) parent value in the output data table will not represent a child value

that was not generalized to the same parent value. For example, in Figure 2.1, if $Writer$ is generalized to $Artist$, then an instance of $Artist$ in the output data table does not cover $Painter$ as its child value.

Cell generalization [75] [127] [135] is a local generalization approach that provides flexibility by generalizing some values in their records while keeping the other values in the rest of the records ungeneralized. Local generalization achieves less information loss than global generalization, yet the former method suffers from significant limitations. Existing statistical tools, such as SAS and SPSS, are unable to handle data anonymized by local generalization due to the complexity of performing analysis on overlapping subdomains [131]. Furthermore, even though sibling nodes are not affected by locally generalizing one value to another, e.g., $Painter$ to $Artist$ in Figure 2.1, most standard data mining methods treat $Painter$ and $Artist$ as two independent values, which is not the case [44]. For instance, mining classification rules may create fuzzy rules; the following two rules make it ambiguous to classify a new $Painter$: $Painter \rightarrow class1$ and $Artist \rightarrow class2$.

All the above generalization schemes choose a *single* value for generalization at a time. This type of generalization can be described as *single-dimensional*. *Multidimensional generalization* [76] [77], on the other hand, strives to reduce information loss by considering a *vector of values*, instead. For example, in Table 2.1(a), $Painter$ in record $\langle 1951, Male, Painter\rangle$ can be generalized to $Artist$, but $Painter$ in record $\langle 1961, Female, Painter\rangle$ can be generalized to $Working$. This is different from local generalization because the choice of generalizing $Painter$ in the above example is dependant upon the other values in the record, i.e., $\langle YoB, Male/Female\rangle$.

The idea is to divide the multidimensional domain space of an input data table into non-overlapping generalization regions. Every region contains a set of generalized records from the table, where every raw record is uniquely mapped to its corresponding region. By that intuition, a region can be considered as a QID group because every region contains a disjoint subset of generalized records.

2.3.3 Perturbation

Perturbation [2] [91] is an anonymization technique that replaces original data with synthetically generated data without significantly distorting the original statistical information. Generally speaking, this method is useful for publishing anonymous aggregate statistics. Next, we describe two perturbation techniques for publishing *differentially-private* data.

Suppose there exists a function f that maps a dataset D to real values. The *sensitivity* [37] of f is the maximum change in the true answer due to adding or removing a single record in D. For example, suppose f answers to count queries over D. The maximum change of a true query answer due to adding/removing one record in D is 1. Therefore, the sensitivity of f in this case is 1. The sensitivity of f, symbolized as Δf, is defined as follows:

Definition 2.2 (*Sensitivity*)**.** For any function $f : D \rightarrow \mathbb{R}^d$, the sensitivity of f is

$$\Delta f = max_{D,D'} ||f(D) - f(D')||_1 \tag{2.3}$$

for all D, D' differing by one and only one record. ■

The literature has defined two techniques to aid in realizing differential privacy: the *Laplace mechanism* [37] and the *exponential mechanism* [94].

The **Laplace mechanism** first computes the true answer of a function f over a dataset D, $f(D)$, and then adds to $f(D)$ a noise drawn from the Laplace distribution. More formally, the Laplacian noisy answer given by the Laplace mechanism is $f(\hat{D}) = f(D) + \texttt{Lap}(\lambda)$, where $\texttt{Lap}(\lambda)$ is a noise drawn from the Laplace distribution with probability density function $\Pr(x|\lambda) = \frac{1}{2\lambda}\texttt{exp}(-|x|/\lambda)$ of variance $2\lambda^2$ and mean 0.

Theorem 2.1. *[37] Given any function $f : D \rightarrow \mathbb{R}^d$ over an arbitrary domain of database D with d attributes, an algorithm Ag that adds independently generated*

noise with distribution $\texttt{Lap}(\Delta f/\epsilon)$ *to each of the* d *outputs satisfies* ϵ*-differential privacy.* ■

For example, suppose f answers to count queries over D. Given a privacy budget ϵ and the sensitivity of f, Δf, then according to Theorem 2.1, $f(\hat{D}) = f(D) + \texttt{Lap}(1/\epsilon)$ satisfies ϵ-differential privacy.

As for the **exponential mechanism**, it is used in situations where the true answer is not a real value. In this case, the exponential mechanism assigns a probability to every candidate output o in the output domain $\mathcal{O}$. The assigned probability is based on a utility function u that gives real-valued scores to every candidate output $o \in \mathcal{O}$. Outputs with higher scores are exponentially more likely to be selected by the exponential mechanism. This ensures that the selected output is close to the true output.

Theorem 2.2. *[94] Given any utility function* $u : (D \times \mathcal{P}) \to \mathbb{R}$ *with sensitivity* $\Delta u = max_{\forall p,D,D'} |u(D,p) - u(D',p)|$*, an algorithm Ag that chooses an output* p *with probability proportional to* $\texttt{exp}(\frac{\epsilon u(D,p)}{2\Delta u})$ *satisfies* ϵ*-differential privacy.* ■

Chapter 3

Literature Review

In recent years, we have been witnessing a continuous expansion in information technology that facilitates our daily lives. Smart phones, GPS-equipped devices, smart card automated fare collection (SCAFC) systems, and sensory technology are but few examples of how pervasive information technology has become. This technological burst, albeit advantageous, requires collecting users' information (e.g., name and date of birth for registration purposes) or data (e.g., current location in order to establish daily habits for customized services). However, collecting data about individuals comes at the cost of compromising their privacy, whether by poor handling of such collected data or risking direct privacy attacks. Such risk stems from the need to publish the collected data to third parties for various reasons such as academic research, information extraction, or general analysis for service improvement.

In this chapter, we explore prominent works that have been proposed for protecting individuals' privacy in the domain of privacy-preserving data publishing [44]. Particularly, we present an in-depth literature review of existing anonymization methods for four types of data: data streams, trajectory data, transaction data, and relational data, each of which has shown to be susceptible to exposing individuals' sensitive information.

3.1 Data Streams

A data stream, or streaming data, can be described as a source of "live" data being generated *continuously* and on the fly. Examples of data streams include live click streams, sensory data, stock market data, and network traffic. The collected data offers a tremendous opportunity for extracting interesting knowledge [46]. For instance, mining network traffic can provide detection/prevention mechanisms against network attacks. Moreover, a situation may arise, such as in military applications, that demands analyzing streaming data on the fly in order to provide proper responses. Data streams can also be used for live monitoring of moving individuals thanks to the pervasiveness of location-aware devices that constantly report spatio-temporal data about the individuals carrying such devices.

Mining automatically-collected live data poses a potential privacy risk against the individuals whose data is being streamed. For example, if a mailing company decided to monitor its truck drivers' driving routes for improving mail delivery services, the employees may not want to expose certain sensitive locations that are not related to their job, such as hospitals or pharmacies [92]. Consequently, there is a dire need to anonymize data streams while preserving their utility for various data mining tasks.

Anonymizing continuously generated data is a challenging task. Data streams are characterized by being time-variant and potentially infinite; therefore, it is inconvenient to store all the generated data in conventional databases. Another challenge in mining such data is the computational complexity due to the fact that data streams are continuous, transient, and time-variant [53]. This thesis presents the first work to anonymize *trajectory streams*. We aim at achieving anonymous streams that (1) preserve data truthfulness with comparison to the collected data in the raw stream and (2) provide accurate data mining results. We next review some of the prominent works in the area of anonymizing data streams.

Li et al. [81] targeted preserving individuals' privacy in numerical data streams.

Dwork et al. [38] proposed a set of algorithms based on differential privacy, which produces noisy answers, to address some *specific* counting tasks. Chan et al. [24] addressed the problem of privacy-preserving aggregation on sensitive streams. In this setting, an untrusted aggregator should not learn the exact points of the moving individuals; but rather, only estimated aggregate statistics. This goal is different from ours: the work in [24] achieves statistic aggregations while our work preserves the exact whereabouts of each moving individual, allowing data collectors to provide customized services, e.g., recommendations, tailored for each individual. Hence, we aim at supporting a wider *range* of operations on the output anonymous data.

Zhou et al. [144] proposed a framework for k-anonymizing a stream of relational data. Clusters are created and filled with arriving data elements. Once a cluster contains enough data elements that belong to at least k moving individuals, the data is published at the same generalization level. To limit information loss due to generalization, both [144] and [142] incorporate a prediction mechanism of future data elements. Another approach targeting the same type of streaming data is *CASTLE* [22], a cluster-based approach. It incorporates cluster merging and splitting mechanisms based on a maximum allowable delay parameter. In *SANATOMY*, Wang et al. [123] employed anatomy to publish an ℓ-diverse data stream. They also assumed the arrival of a single data element in the stream at any given timestamp. Li et al. [83] proposed a method called *SKY* that allows a data owner to determine how much an anonymous stream deviates from its raw version. All the works in [22] [123] [83] enforce a pre-defined time constraint to limit the delay of the published data. We argue that imposing time constraint damages data freshness in the stream. Furthermore, nearly-expiring data elements that do not yet meet the privacy requirements are generalized to higher levels, adversely affecting the utility of the anonymous data. Unlike the work presented in this thesis, which anonymizes streams of trajectories, all the above works target streams of relational data. Moreover, the imposed privacy models in the above works is heavily based on

the traditional k-anonymity model, which does not accommodate attribute linkage attacks. Whereas, the work in this thesis thwarts both attribute and record linkage attacks.

In a closely related research area, researchers have studied and proposed several approaches for anonymizing *continuous data*. In the context of continuous data release, updated versions of a data table are published on a regular basis, e.g., every week [21] [129] [122]. Wang et al. [122] proposed a method for anonymizing temporal data in relational format. Their method is based on temporal record relocation within a window of multiple releases. Xiao and Tao [129] considered the problem of re-publishing updated relational data tables. Their method, called *m-invariance*, is the first to address the insertion and deletion of records in the updated data table. If a record does not meet the imposed privacy requirement, counterfeit records are created in order to achieve an m-invariant updated table. Applying m-invariance to trajectory streams is not suitable due to the following reasons. First, m-invariance anonymizes relational data. Trajectory data is high-dimensional by nature; thus, applying methods based on QID attributes incurs significant data loss. Second, m-invariance does not assume the existence of data streams, taking advantage of not having a sharp time constraint for publishing the updated data table. In contrast, our proposed method in this thesis maintains the transient nature of streams by anonymizing and publishing newly-arrived data on the fly. Third, m-invariance is achieved by adding counterfeit records to the data table. On the other hand, our method maintains truthfulness because all published records belong to real-life moving individuals. This property gives more credible results when analyzing the anonymous data. In summary, continuous data release does not require publishing the data table at the time of data collection since the table does not contain live data. Algorithms for anonymizing continuous data are not suitable for potentially infinite streams of transient and time-critical data because these properties require dynamic and scalable processing with little time delay.

3.2 Trajectory Data

Trajectory data is a spatio-temporal data that contains a location dimension and a time dimension. A single trajectory is the trace generated by a single individual (or a moving object), where every visited location is coupled with a timestamp. Timestamps within a single trajectory are non-decreasing and are drawn from a timestamp universe, whereas a location may appear multiple times and/or consecutively.

Mining trajectory data has various applications [51], such as traffic analysis, city infrastructure planning, and customized services provided through analyzing human behavior [88]. Such significance has demanded publishing collected trajectories for data analysis. However, publishing trajectory data without adequate anonymization can put the privacy of the moving individuals at risk. For instance, if a target victim takes the same path almost every morning, an attacker can infer that the starting point is the victim's home and the ending point is the victim's workplace [1]. Trajectory data contains a time dimension that renders the data high-dimensional and, in most cases, extremely sparse. Handling sparse data is a challenging problem because data points are scattered in the high-dimensional space, causing excessive data distortion by the anonymization algorithm, and, consequently, resulting in poor data utility. There has been extensive work on privacy-preserving trajectory publishing under different assumptions and privacy models [17]. Next, we categorize these works based on their privacy models, namely, *syntactic* and *semantic*.

Syntactic privacy models, such as k-anonymity [115] and ℓ-diversity [90], stipulate that the output dataset of an anonymization algorithm must adhere to some syntactic conditions in order to protect data records and sensitive items. Nergiz et al. [103] were the first to apply k-anonymity to trajectory data, whereby every trajectory in its *entirety* must be indistinguishable from at least $k-1$ other trajectories. Abul et al. [1] proposed (k, δ)-*anonymity* that enforces *space translation*, resulting in having every trajectory coexisting with a minimum of $k-1$ other trajectories within

a proximity of δ. Monreale et al. [99] achieved k-anonymity by using *spatial generalization*. The novelty of their method lies in dynamically generating geographical areas based on the input dataset, as opposed to generating a fixed grid [139]. Hu et al. [67] applied k-anonymity to a trajectory dataset with respect to a reference dataset containing sensitive events. Particularly, they developed *local enlargement* that transforms the trajectory dataset such that every sensitive event is shared by at least k users. Pensa et al. [104] studied the problem of anonymizing sequential data by preserving frequent sequential patterns. The authors consider temporal sequentiality, which can be considered a simpler form of trajectory data. To account for high dimensionality in sequential data, the authors use a prefix tree to structure sequences of temporal items. Their proposed method is based on k-anonymity and, thus, thwarts only identity linkage attacks.

In addition to generalization [103] [99] [139] [112] and space translation [1] [104], suppression-based techniques [117] [27] [30] have been proposed to achieve k-anonymity-based privacy models. Terrovitis and Mamoulis [117] developed a privacy model that assumes different adversaries possess different background knowledge, and, consequently, they modeled such knowledge as a set of projections over a sequential (trajectory) dataset. Their anonymization method limits the inference confidence of locations to a pre-defined threshold. Similarly, Cicek et al. [30] ensured location diversity by proposing *p-confidentiality*, which limits the probability of visiting a sensitive location to p. *Local suppression* has been used in [27] [49] to boost data utility. Under local suppression, only *some* instances of an item will be removed from the dataset - as opposed to *global suppression*, which removes *all* instances in the underlying dataset.

All of the above techniques incorporate *syntactic* privacy models, which have been proven to be prone to several privacy attacks [126] [48] [72]. Moreover, due to the curse of high dimensionality [3], applying anonymization methods that project the concept of QID attributes [84] [89] onto trajectory data imposes significant

data loss because every doublet of location and timestamp in a trajectory is considered to be a distinct QID attribute. In other words, the notion of a fixed set of QID attributes in trajectory data does not exist anymore because of the continuously changing reported locations and timestamps [139]. For this reason, this thesis presents a trajectory sanitization algorithm under a *semantic* privacy model, namely differential privacy.

Protecting trajectories under differential privacy [35] has been gaining increasing attention in the past few years. Some of these works focus on publishing data mining results, e.g., mining trajectories for frequent location patterns [64] [65], whereas other works aim at publishing differentially-private trajectories. We focus on the latter approach as it provides more analytical power to data recipients, and it is more related to the work presented in this thesis. For more information on the interactive setting, non-interactive setting, and recent works on differentially-private data publishing, we refer the reader to [36] [79] [120], respectively.

Chen et al. [26] introduced the first differentially-private work to publish large-volume *sequential locations*. Although their sanitization algorithm preserves count queries and frequent sequential pattern mining [5] only, data recipients can perform several other data mining tasks on the sanitized output dataset. In a more recent study, He et al. [61] proposed to synthesize trajectories from a probabilistic model based on the *hierarchical reference system* of the input dataset. Xiao and Xiong [132] considered temporal correlation to protect true locations *within a single trajectory*, as opposed to *user-level* privacy (adopted in the work presented in this thesis), which protects the presence of an entire trajectory *within a dataset*. Xiao and Xiong's temporal correlation technique is achieved by hiding the true location within a set of probable locations, called *δ-location set*. Works similar to [26] [61] [132] define trajectories as *sequential locations*. We argue that in real-life trajectories every location is paired with a timestamp that should also be accounted for by the trajectory publishing mechanism. For example, it is important to know busy streets

when performing traffic analysis, but it is equally important to also know the time period during which traffic jams peak. Therefore, this thesis defines trajectories as a sequence of locations *coupled* with timestamps.

Jiang et al. [69] sampled distance and angle between true locations within a trajectory in order to publish an ϵ-differentially private version of that trajectory. However, their method publishes a single trajectory only, i.e., the entire privacy budget ϵ is spent on sanitizing a single trajectory. Primault et al. [105] proposed to hide moving individuals' points of interest [47], such as home or work. While their method protects against inference attacks, we argue that hiding points of interest is harmful for applications that rely on such information, e.g., traffic analysis and probabilistic flowgraph analysis. In this thesis, we present a trajectory sanitization method that aims at maintaining the spatio-temporal characteristics of the raw trajectories in order to support a wide range of data analysis tasks. Assam et al. [9] presented a method whereby both spatial and temporal domains are sanitized and published under differential privacy. In [9], trajectories are represented by a series of GPS-like data points (x, y, t). Their method first creates temporal blocks (called *Running Windows*) that average all the data points that fall in them. Every *Running Window* is then represented by its average location and timestamp values, which are further perturbed under the Laplace mechanism. The sequence of noisy averages constitutes the sanitized trajectory. Assam et al.'s method is robust enough to output a *single* trajectory with fairly good utility. However, it is unclear how their method can handle multiple moving individuals since the output of their proposed algorithm is always a single sequence of noisy averaged data points. In contrast, this thesis presents a trajectory sanitization algorithm that outputs a *multiset* of trajectories, each belonging to a unique moving individual.

3.3 Transaction Data

Transaction datasets organize a set of *items* pertaining to each individual in the dataset [119] [63] [118]. Each record contains an arbitrary number of items drawn from the dataset's universe of items. Examples of transaction data include click streams, query logs, and lists of purchases. Such data is considered a rich source from which researchers can learn about individuals' habits for various purposes, such as advertising [58].

Even though this thesis does not present solutions for anonymizing transaction data, per se, this type of data is high-dimensional by nature; each item can be considered an independent dimension and a potential piece of an attacker's background knowledge. In other words, each transaction item is considered a QID attribute, making anonymizing high-dimensional transaction data a challenging problem that may not be easily solved by applying traditional privacy models. This section discusses prominent existing techniques in the literature for anonymizing transaction data. We broadly divide these techniques based on the type of their privacy model, i.e., syntactic or semantic.

Terrovitis et al. [118] argued that an attacker requires an unrealistic effort to acquire background knowledge covering the entire domain of items (QID attributes) in a transaction dataset. Thus, they restricted the attacker's knowledge to a maximum of m items and proposed k^m-*anonymity*. This privacy model extends k-anonymity by hiding any transaction with at most m items in a group of at least $k-1$ other indistinguishable transactions. To achieve anonymity, Terrovitis et al. [118] used global generalization, which maps *all* occurrences of the same value in the dataset to a more general value according to a given domain hierarchy. Terrovitis et al. [119] enforced k^m-anonymity using local (or cell) generalization, which may not necessarily generalize all the occurrences of a value, thus improving the anonymous data utility.

k^m-anonymity does not account for the existence of sensitive attributes in

transaction data. The works in [50] [137] [138] studied the case where the universe of items is considered public knowledge, and a transaction dataset also includes dedicated attributes containing sensitive information about individuals in the dataset. Xu et al. [138] proposed a flexible privacy model called (h, k, p)-*coherence*, which restricts the attacker's knowledge to a maximum of p items, the size of any indistinguishable group of transactions (containing at most p items) to at least k, and the probability of inferring a sensitive attribute from any group of indistinguishable transactions to at most h. Xu et al. [137] further explored preserving *frequent itemsets* in an (h, k, p)-coherent dataset. Furthermore, they made a significant improvement on scalability by introducing a *border-based* representation of moles and nuggets, which can grow exponentially in number. Both [137] and [138] use global suppression to enforce (h, k, p)-coherence. In global suppression, if an item is chosen for suppression, then all occurrences of that item in the dataset are suppressed (removed), as well. Ghinita et al. [50] proposed a permutation-based method by which transactions with similar items are grouped together, and inferring a sensitive item is bound to a pre-defined threshold. In order to enhance the anonymous data utility, the authors in [52] proposed a general framework that models the problem of anonymizing transaction data as a clustering problem. To enhance data utility, the authors further devised two algorithms by which both privacy and user-specified utility requirements are met in the anonymous data, respectively.

Differential privacy has also been integrated in transaction data anonymization algorithms [29] [66] [140] [85] [74]. Chen et al. [29] were the first to propose a differentially-privacy solution in the context of publishing differentially-private transaction data for data mining purposes. They proposed a top-down partitioning approach based on a *context-free* taxonomy tree. Data utility is defined in terms of count queries, which is the basis of various data mining tasks. Motivated by the AOL data release incident [14] [58], Hong et al. [66] were the first to address the problem of publishing anonymous search logs that maintain the same input

logs schema. Focusing on maximizing utility, they transformed the problem of finding the maximum-utility differentially-private output into an optimization problem. However, in a preprocessing step, there are two cases in which users' logs are removed from the input dataset: (1) users with unique queries, and (2) users with logs that result in changing the optimal solution by more than some threshold. Additionally, Hong et al.'s algorithm outputs synthetically-generated logs (as opposed to perturbed input logs). We argue that, while data utility is maximized for aggregate statistics, maintaining information at the log level is more beneficial to data truthfulness.

3.4 Relational Data

Generally speaking, a relational data table consists of rows and columns: each row represents a unique individual and each column represents an attribute that describes a certain type of information about individuals in the table. Some of these attributes, known as *quasi-identifiers* (QID), can be used to re-identify individuals even in the absence of identifying pieces of information in the table. A decent amount of work has been done towards anonymizing relational data. One notable proposition is k-anonymity [110] [111] [115]. Initially proposed by Samarati and Sweeny [111], k-anonymity stipulates that each record in a relational data table must be indistinguishable from at least $k-1$ other records over the QID attributes. Hence, a successful record linkage attack is bound by $1/k$.

Achieving *optimal* k-anonymity has been proven to be NP-hard [75]. Optimal anonymity is achieved when the anonymous table has the least degree of anonymity yet is most informative compared to all other possible anonymous tables produced from enforcing the same privacy requirement. LeFevre et al. [75] proposed *Incognito* to achieve optimality by enforcing *full-domain generalization* over the QID attributes. Under full-domain generalization, if a certain value from a QID attribute

is generalized to a higher level in a given taxonomy tree, then all the other domain values from the same attribute are generalized to the same hierarchical level, as well. *Incognito* achieves optimal k-anonymity but runs exponentially with respect to the number of QID attributes.

In the presence of attributes with sensitive information, a new type of privacy attack arises that targets associating sensitive information with certain individuals. This type of privacy attack is called attribute linkage, which k-anonymity fails to prevent. To thwart attribute linkage attacks, Machanavajjhala et al. [89] introduced the concept of *ℓ-diversity*, which stipulates that every group of indistinguishable records is required to have at least ℓ "well-represented" records w.r.t. the sensitive values. The simplest interpretation of "well-represented" is for every group to have a minimum of ℓ sensitive values. Wang et al. [121] proposed a more flexible privacy notion called *confidence bounding* that defines a separate privacy template (requirement) for each sensitive value. Given the set of templates, a sensitive value within a group of indistinguishable records has a frequency that does not exceed the confidence threshold defined by the associated template for that sensitive value. Li et al. [84] proposed *t-Closeness*, which requires the distribution of a sensitive value in a group of indistinguishable records to be close to the distribution of the same sensitive value in the entire table. t-Closeness is particularly beneficial when some sensitive values naturally occur more frequently than others. For example, in a patients' data table, *Allergy* is more likely to appear in the *Disease* attribute than *Cancer*. Wong et al. [127] combined both k-anonymity and confidence bounding in their proposed *(α, k)-Anonymity* model, which requires every record to be shared by at least k other records, and the confidence of inferring any sensitive value in a group of indistinguishable records to be $\leq \alpha$.

Several privacy-preserving algorithms have been proposed for publishing anonymous relational data tables for the goal of classification analysis [125]. Iyengar [68] was the first to address this issue by devising a genetic algorithm, which proved to be

costly. In his work, classification accuracy on training data was measured using his proposed *classification metric*, which was later used by Bayardo and Agrawal [15]. Fung et al. [45] proposed Top-Down Specialization (TDS), a heuristic approach by which a relatively accurate classifier can be built based on the anonymous data. The approaches in [68] [15] [45] achieve anonymity by enforcing *single-dimensional* generalization. LeFevre et al. [76] [78] noticed that it is possible to improve data utility, and thus classification accuracy, by employing *multidimensional* generalization, and thus they proposed their anonymization method *Mondrian*. This thesis uses the latter generalization technique for anonymizing relational data.

All the aforementioned works use syntactic privacy models that extend k-anonymity. Such privacy models rely on estimating the attacker's background knowledge and, thus, are susceptible to syntactic-based privacy attacks, such as *minimality attack* [126], *composition attack* [48], and *deFinetti attack* [72]. Therefore, we adopt differential privacy as the privacy model for anonymizing relational data in this thesis.

ϵ-Differential privacy is a rigorous privacy notion that negates the impact of the attacker's background knowledge on privacy risks. Thus, the literature has been shifting towards adopting this semantic privacy model for data publishing. Differential privacy has two settings, namely *interactive* and *non-interactive*. Next, we present some of the most relevant work in the literature to tackle publishing relational data under differential privacy.

Releasing private histograms [25] [60] [80] [134] [133] is a related field of research to privacy-preserving data publishing. A histogram is a set of disjoint regions containing data points over the domain of a dataset. Hay et al. [60] proposed a method for releasing more accurate private histograms under differential privacy; however, their work is limited to single-dimensional histograms as in [16] [136]. Although [60] and [130] provide noise optimization for *range* queries, Li et al. [80] enhanced their work by achieving optimal noise variance for a variety of *workload*

queries. Xiao et al. [134] proposed a method by which multidimensional partitioning was used to release differentially-private histograms.

All the techniques in [60] [130] [80] [128] [134] [133] are based on the *interactive* setting that requires the queries to be provided in advance. It is worth noting that in the interactive mode it is possible to synthesize a noisy contingency table of the underlying dataset D by issuing a set of queries spanning all combinations of domain values [133]. We argue that if D is sparse with a large domain, then performing operations on the noisy contingency table results in poor data utility. This is because sparse data points tend to be scattered over the domain space resulting in extremely small counts in most subdomains. Thus, raw counts will be outweighed by the added noise, rendering a query answer useless. This situation worsens when multiple users query D with a common ϵ [134]. If a user is assigned a small fraction of ϵ, more noise will be added to the true query answer. Once all ϵ is consumed by the queries, the data holder has to shut down the database entirely. Therefore, the work presented in this thesis employs differential privacy in the *non-interactive* setting.

Several methods have been proposed for publishing a private contingency table [13] [33] [107] [130] [31]. Barak et al. [13] used linear programming to post-process a differentially-private output in order to publish a set of consistently integral marginals of a contingency table. Though it guarantees differential privacy, Barak et al.'s work does not improve accuracy in the output data. A similar problem has also been studied by Ding et al. [33] and Qardaji et al. [107]. Xiao et al. [130] succeeded in improving the accuracy of a differentially-private contingency table by proposing *Privelet*, a method based on wavelet transformation on the data attributes. On the same note, Cormode et al. [31] proposed to optimize the computation required when publishing a contingency table of a sparse dataset, in a differentially-private way. They achieved that by utilizing compact summaries computed *directly* from the input dataset, as opposed to computing a noisy contingency table first, which is costly for sparse data. However, the utility of their private summaries is similar to that of

a generated contingency table. We argue that releasing a private contingency table can be damaging to the accuracy of the analysis as the added noise grows larger for sparse data. Instead, in this thesis we focus on publishing a generalized version of the input data without compromising data utility.

To account for the inherent challenge of releasing sparse or high-dimensional data in a differentially-private way, Zhang et al. [141] proposed a method called *PrivBayes*. In *PrivBayes*, a Bayesian network is utilized to construct a set of low-dimensional subcubes that approximate the joint distribution in order to release a synthetic dataset that in turn approximates the distribution of the high-dimensional input dataset.

In the *non-interactive* setting, the problem of publishing differentially-private relational data has been studied in [96] [106]. The general idea in both works is to partition the input dataset into smaller groups of "similar" records, then release a noisy count of the records in each resultant partition/region. Mohammed et al. [96] proposed an algorithm called *DiffGen*, a top-down specialization approach that aims at producing a generalized version of the input data in a differentially-private setting. *DiffGen* uses a single-dimensional partitioning strategy that greedily chooses a split attribute that maximizes the utility of the output dataset, without violating differential privacy. When an attribute is chosen, a split is performed in accordance with the hierarchy of the domain values dictated by an input taxonomy tree. Mohammed et al. improved classification accuracy when they experimentally compared with *DiffP-C4.5* [41], an interactive approach for classification analysis.

In [106], the authors proposed a general framework that can be instantiated differently based on the desired implementation. They proposed a meta-algorithm, called *RPS*, that takes into consideration the distribution of the data points (records) in the multidimensional space R of the dataset. The meta-algorithm recursively performs *binary* partitioning over R to achieve nearly balanced regions. A median point is computed in a differentially-private way from the domain of the chosen attribute

to produce two non-overlapping regions. Noisy counts of data points in each resultant region are then returned to compose the overall sanitized dataset. *RPS* has two weaknesses. First, for the case of relational data, *RPS* randomly chooses a split attribute [106], as opposed to carefully choosing a split attribute that would result in better utility depending on the desired utility measure (adopted in this thesis). Second, even though *RPS* is theoretically capable of choosing a split point that spans across multiple dimensions (attributes), it does not scale for datasets with a large number of dimensions because it inefficiently considers all combinations of values across all dimensions [108]. More specifically, in every iteration, *RPS* chooses one split point from $\Omega(A_1) \times \ldots \times \Omega(A_d)$ possible combinations, where d is the number of attributes and $\Omega(A_i)$ is the domain of attribute A_i in a given region.

Chapter 4

Anonymizing Trajectory Streams

4.1 Introduction

The improvement of information technology in the past years has facilitated sharing data among organizations, firms, and to the public. Location-aware devices, such as GPS and mobile phones, *constantly* report spatio-temporal data of a moving object or the individual carrying this object. In many cases, it is important to publish the automatically-collected data on the fly for various purposes, such as traffic analysis, live monitoring of moving objects, and mining recent events in a data stream. This process becomes of vital importance, especially when it is essential to take immediate actions or follow certain detection or prevention measures. Nevertheless, releasing the automatically-collected raw data by a data holder for analysis and service improvement may compromise individuals' privacy from whom the data is being collected. We assume that part of the recipients of a published data stream are untrustworthy, and they may attempt to identify target victims or infer their sensitive information. In this chapter, we study the challenges in anonymizing a stream of trajectories, and propose an efficient algorithm to anonymize a trajectory stream with the goal of minimizing data distortion.

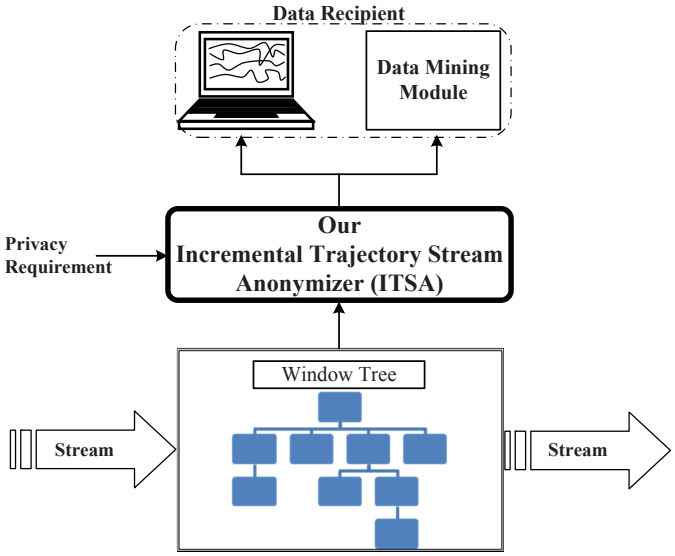


Figure 4.1: Mining trajectory stream over a sliding window

Figure 4.1 shows an overview of the trajectory stream environment. A trajectory stream S is a continuous sequence of *triples*, in which each *triple* has the form $\langle ID, loc, t\rangle$, indicating that a moving individual ρ with identifier ID is at location loc at timestamp t. A combination of loc and t is called a *doublet*. We assume that the trajectory stream S is published for stream mining [46] or simply for the purpose of displaying the trajectory paths on screen.

We propose a trajectory stream anonymization method based on a *sliding window* [53] [11]. The literature has defined two types of sliding windows: *count-based* and *time-based* [12] [53]. The former type defines a window that includes the N most recent data elements, while the latter type defines a window that includes all the elements that belong to the most recent N timestamps. We adopt a time-based sliding window because it is a more general representation of a count-based window. However, our method can seamlessly accommodate a count-based window with no impact on the general approach. Hence, our approach models data stream as a sequence of sliding windows in which the most recent window includes the triples having the most recent N timestamps.

Table 4.1: Raw sliding windows, $W_{1\to3}$ and $W_{2\to4}$, on trajectory stream

ID	**Timestamps**				SA
	1	2	3	4	
1		b.2	c.3	d.4	$\boldsymbol{sen\_val_1}$
2	a.1	f.2	c.3	d.4	sen_val_2
3		b.2	c.3	d.4	sen_val_3
4	a.1	f.2	c.3		sen_val_4
5		b.2	c.3		sen_val_5
6			c.3		sen_val_2
7		f.2		d.4	sen_val_3
8			c.3	e.4	$\boldsymbol{sen\_val_1}$
	$W_{1\to3}$				
		$W_{2\to4}$			

The Copenhagen International Airport is testing a mechanism for monitoring travelers' movements in real-time by following their Wi-Fi trails with the goals of improving airport design and security, directing the flow of travelers, and providing customized services to travelers [102]. Yet, disclosing the raw trajectory stream to some third-party service provider, such as an airline company or an outsourced security firm, may compromise the travelers' privacy.

The following example illustrates two types of privacy attacks that an adversary can carry out by having access to the data stream. We clarify three points about Example 4.1.1 and any of its continuations throughout the rest of this chapter. First, Example 4.1.1 uses the English alphabet to represent locations, where each letter represents a distinct location. No specific order exists between locations, e.g., location b may appear before location a. Third, location symbols in this example are not associated with any symbol or notation used outside the context of this example or any of its continuations.

Example 4.1.1. Table 4.1 shows the trajectories of eight travelers sorted by their IDs. Table 4.1 includes the raw data and Table 4.2 includes the anoynmous data.

Table 4.2: Anonymous sliding window $\hat{W}_{2\to4}$ for $L = 2$, $K = 2$, $C = 40\%$

ID	**Timestamps**			SA
	2	3	4	
1		c.3	d.4	$\boldsymbol{sen\_val_1}$
2	f.2	c.3	d.4	sen_val_2
3		c.3	d.4	sen_val_3
4	f.2	c.3		sen_val_4
5		c.3		sen_val_5
6		c.3		sen_val_2
7	f.2		d.4	sen_val_3
8		c.3		$\boldsymbol{sen\_val_1}$

For simplicity, this example considers timestamps 1-4; however, in reality, timestamps continue indefinitely. Let us assume that sensitive information is being collected from travelers along with trajectories. The sensitive information is displayed in the sensitive attribute SA. A potential sensitive attribute could be *Disability* where travelers with *Epilepsy*, for instance, may require special attention to facilitate their journey. The data holder (the airport) can specify a set of sensitive values from the sensitive attributes. Upon publishing the anonymous data, sensitive values should not be associated with their corresponding travelers. Suppose sen_val_1 is the only sensitive value in SA in this example.

Let the size of the sliding window be $N = 3$. The first window $W_{1\to3}$ includes doublets with timestamps 1-3, as indicated by the dashed box in Table 4.1. As the window slides with step size = 1, the second window $W_{2\to4}$ now includes doublets with timestamps 2-4 with no traces of doublets having timestamp 1. We note that the absence of doublets within a given window (the empty spots in Tables 4.1 and 4.2) indicates no change in a traveler's location.

Suppose an adversary has access to the trajectory stream in the form of a sliding window, as in Table 4.1. It is possible to identify a target victim's trajectory and/or sensitive value by performing the following privacy attacks.

Identity linkage, also called *record linkage*, takes place when the collected trajectories contain a sequence of doublets with a rare appearance. This allows an

adversary to uniquely identify a target victim. For example, suppose that the current window is $W_{2\rightarrow4}$, and that an adversary knows that a target victim has visited location e at timestamp 4. $W_{2\rightarrow4}$ contains *only one* trajectory ($ID = 8$) with doublet $e.4$. Hence, the adversary is able to learn the victim's other visited locations and sensitive value.

Attribute linkage takes place if there is a group of records, sharing the same sequence of doublets, that contains infrequent sensitive values. These values can be associated with their pertinent individuals with high confidence. This type of privacy attacks is also known as homogeneity attack [84] [89]. Suppose that an adversary knows that a target victim has visited locations b and d at timestamps 2 and 4, respectively. $W_{2\rightarrow4}$ shows that one of two records that contain $\langle b.2 \rightarrow d.4\rangle$ has the sensitive value sen_val_1. Hence, the adversary is able to infer that the target victim has sen_val_1 with 50% confidence. ■

Challenges. Data streams are characterized by being time-variant and potentially infinite. Therefore, it is infeasible to first store the data and then anonymize it. Rather, the streaming data has to be anonymized on the fly. Moreover, every possible combination of location and timestamp in trajectory data forms a distinct dimension [137]. This characteristic is referred to as *the curse of high dimensionality* [3]. For example, if the airport in Example4.1.1 contains 200 hotspots (access points), then monitoring travelers' movements over the period of 60 minutes will result in 12,000 dimensions. Consequently, applying anonymization methods based on quasi-identifier attributes [115] will impose excessive data loss, rendering the anonymous data useless for any data analysis.

Contributions. To the best of our knowledge, this is the first work to anonymize high-dimensional trajectory stream [6]. We summarize our contributions in this chapter as follows. First, to address the transient nature of trajectory streams, we propose an anonymization method based on a *dynamically* updated sliding window, where trajectories are modeled as a prefix tree to ensure compactness and

efficient data retrieval. Second, a naive solution to anonymizing trajectory streams is by simply anonymizing every single window independently. To avoid this redundancy, we identify some important properties in trajectory streams, and utilize these properties to efficiently anonymize trajectories in a sliding window by *incrementally* updating the prefix tree. Third, our proposed method guarantees that the output anonymous trajectory stream satisfies LKC-privacy [97] [98] [43]. LKC-privacy is a flexible privacy model that has proven efficient in handling high-dimensional data [98]. Fourth, our experimental evaluation on simulated and real-life datasets demonstrates that our proposed algorithm is capable of handling large-volume trajectory streams without compromising their utility.

4.2 Problem Definition

A data holder is constantly collecting the trajectories of some group of moving individuals. A trajectory tr is a sequence of *triples*. A triple $\langle ID_\rho, loc, t\rangle$ that belongs to individual $\rho \in Popoulation$ reports ρ's location $loc \in \mathcal{L}$ at timestamp $t \in \mathbb{N}$, where $Popoulation$ is the universe of moving individuals generating trajectories and $\mathcal{L}$ is the universe of all possible locations. We define a trajectory stream as follows.

Definition 4.1 (Trajectory stream)**.** A trajectory stream $S = \{\langle ID_1, loc_{ID_1}, t_{ID_1}\rangle, \langle ID_2, loc_{ID_2}, t_{ID_2}\rangle, \ldots, \langle ID_\rho, loc_{ID_\rho}, t_{ID_\rho}\rangle, \ldots\}$ is a continuous sequence of triples generated by every moving individual $\rho \in Population$. ■

Assumptions. We assume that triples are being generated continuously; therefore, it may not be feasible to store all the data in a conventional database. Moreover, we assume that the data recipient is more concerned with *recent* data rather than *outdated* data. In Example 4.1.1, the airport manager may want to monitor the data in real-time by checking for congestions at any gate for the past 60 minutes and react by opening new gates or allocating additional staff. If the airport, however, is to store trajectories for later analysis, say on a weekly or monthly

basis, several anonymization solutions exist for this particular problem [17]. Furthermore, in this work we assume that locations are not considered to be sensitive information. For works related to sensitive locations, we refer the reader to existing solutions [117]. Lastly, we make the assumption that there exists a set of sensitive attributes $SA_1, \ldots, SA_m$ that contain sensitive pieces of information $sen\_val_i \in SA_i$ about each moving individuals.

For the aforementioned reasons, we use a time-based sliding window W to represent the most recent data in a trajectory stream S. A data holder specifies the size of W in terms of timestamps.

Definition 4.2 (Sliding window)**.** Let N be the size of a sliding window W, x be the starting timestamp, and $y = x + N - 1$ be the ending timestamp. $W_{x \to y} = \{triple \in S | x \leq triple.t \leq y\}$, where $triple.t$ denotes the timestamp value in a triple. ■

Following Definition 4.2, Table 4.1 shows $W_{1 \to 3}$, which starts at timestamp 1 and has a size of $N = 3$.

When a window W contains all the proper triples, the next step would be to anonymize this window then publish it - a data recipient has only a view over stream S through a sequence of anonymous windows published one at a time, and any mining operation is performed exclusively on the most recent window. After that, the window *slides*, a process by which *outdated triples* are dropped out and *new triples* are added. When a window slides, it shifts by a certain number of timestamps determined by *step_size*.

Definition 4.3 (Outdated and new triples)**.** From Definition 4.2, given a window size N, $x^+ = x + step\_size$, and $y^+ = y + step\_size$, we define *outdated triples* $O = \{triple \in W_{x \to y} | x \leq triple.t < x^+\}$ and *new triples* $E = \{triple \in S | y < triple.t \leq y^+\}$, where $E \cup W_{x \to y} = \emptyset$. ■

Definition 4.4 (A slide)**.** From Definition 4.3, when window $W_{x \to y}$ experiences a single slide, it becomes $W_{x^+ \to y^+} = (W_{x \to y} - O) \cup E$. ■

Example 4.2.1. Consider Table 4.1. Suppose that the first window is $W_{1\to3}$ (dashed box), $N = 3$, $step\_size = 1$, and that timestamp 4 has not yet appeared. For $W_{1\to3}$, $x = 1$ and $y = 1 + 3 - 1 = 3$. $W_{1\to3}$ is then anonymized and published. At this point, timestamp 4 appears, and since $step\_size = 1$, $O = \{\langle 2, a, 1\rangle, \langle 4, a, 1\rangle\}$ and $E = \{\langle 1, d, 4\rangle, \langle 2, d, 4\rangle, \langle 3, d, 4\rangle, \langle 7, d, 4\rangle, \langle 8, e, 4\rangle\}$. The new window is now $W_{2\to4}$. E is now a subset of $W_{2\to4}$. However, a set union between E and $W_{1\to3}$ remains an empty set because window $W_{1\to3}$ is unaware of any future data; therefore, E does not exist in this particular window. ■

We assume that the data holder publishes the moving individuals' sensitive information along with their trajectories. Therefore, we define an *object table* in which each record corresponds to a unique moving individual ρ and contains ρ's trajectory and sensitive information. More formally,

$$\langle ID_\rho, tr_{x\to y}, sen\_val_1, \ldots, sen\_val_m\rangle,$$

where ID is a record identifier, $tr_{x\to y}$ is ρ's trajectory of doublets corresponding to $W_{x\to y}$, and $sen\_val_i \in SA_i$ are values from sensitive attributes $SA_1, \ldots, SA_m$, respectively, where m is the number of sensitive attributes. For the sake of simplicity, we consider $m = 1$ throughout our examples, i.e., object tables contain only one sensitive attribute. Without loss of generality, our method can handle multiple sensitive attributes. Recall a *doublet* is nothing but *loc* and t from the triple to which it corresponds, we denoted a doublet by *loc.t*. We use this term whenever the focus is rather on trajectories themselves, regardless to whom they belong.

4.2.1 Privacy Threats

A data recipient has access to the most recently updated sliding window W. The published window includes recent moving individuals' trajectories along with their sensitive information. Adversaries are data recipients who attempt to identify a

target victim's trajectory tr and/or sensitive value sen_val. We assume that an adversary possesses a *subsequence* of the victim's trajectory. We denote this subsequence by κ, and we call it the *adversary's background knowledge*. We also assume that κ has a maximum size of L doublets, that is,

$$\kappa = \langle (loc_1.t_1) \rightarrow \ldots \rightarrow (loc_z.t_z) \rangle,$$

where $z \leq L$. We note that given a sliding window with size N, $L \leq N$. κ is a subsequence of a victim's trajectory tr if each and every doublet in κ also exists in tr following the same order.

Obtaining the background knowledge κ from real-time trajectories is feasible due to the following two reasons. First, a relatively long window allows a stalking adversary to gather a considerable amount of data about a target victim. Second, an adversary may learn a victim's trends and habits (e.g., route from home to office), which are highly likely to appear in several windows to come.

The concept of estimating the maximum length of adversary's background knowledge has been previously discussed in the literature [137] [119]. Those works proposed privacy models that take into consideration the attacker's "power". The "power" of any attacker is the maximum number of items known by the attacker about any transaction, in the context of transaction data. We use a privacy model that shares this same concept; i.e., we use L to denote the maximum number of doublets known by the attacker about any moving individual.

It is possible for the data holder to estimate how much background knowledge the attacker can acquire based on the effort needed to obtain such knowledge. If acquiring background knowledge about target victims is deemed relatively easy, then the data holder can increase L to its maximum value. We note that the worst case about setting L is not the entire trajectory size; but rather, the window size N because the size of real-time trajectories is unknown to our anonymization algorithm.

Given an object table T that contains the trajectories of some moving individuals including the target victim's trajectory, κ could be found in a group of trajectories in T. We denote the group of records containing κ by $G(\kappa)$, and the group size, i.e., number of records in $G(\kappa)$, by $|G(\kappa)|$. κ may match only a few records in T. That is, if $|G(\kappa)|$ is very small, then the adversary might be able to uniquely identify the victim's record, thus, learning his/her other visited locations and sensitive value.

Example 4.1.1 demonstrates that given $W_{2\rightarrow 4}$ in Table 4.1, and given that $\kappa = \langle e.4\rangle$, an adversary is able to uniquely identify the victim's record ($ID = 8$) since $|G(\langle e.4\rangle)| = 1$. We refer to this type of attack as *identity linkage*.

If the sensitive values in $G(\kappa)$ are not diverse enough, an adversary might be able to infer the victim's sensitive value *sen_val* with high *confidence*. We denote the probability of inferring the victim's sensitive value *sen_val* from $G(\kappa)$ as follows:

$$Conf(sen\_val|G(\kappa)) = \frac{|G(\kappa \bigcup sen\_val)|}{|G(\kappa)|},$$

where $G(\kappa \bigcup sen\_val)$ is the group of records within $G(\kappa)$ containing both the subsequence κ and the sensitive value *sen_val*. In Example 4.1.1, if $\kappa = \langle b.2 \rightarrow d.4\rangle$, then $Conf(sen\_val_1|G(\langle b.2 \rightarrow d.4\rangle)) = 1/2 = 50\%$.

4.2.2 Privacy Model

We use LKC-privacy model [97] [98] [42] to transform raw window $W_{x\rightarrow y}$ sliding over stream S to an anonymous version $\hat{W}_{x\rightarrow y}$ such that the published object table T thwarts identity and attribute linkage attacks. The reason for choosing LKC-privacy is its flexibility, demonstrated as follows. (a) By changing the input parameters, LKC-privacy can metamorphose into k-anonymity or an instance of ℓ-diversity. This property also implies that if no sensitive information is involved in the process of data publishing, LKC-privacy can still function properly. (b) A larger L provides more protection against adversaries with longer background knowledge.

We note that unlike the work in [62], LKC-privacy does not require the data holder to specify a set of pre-defined subsequences in S. Rather, we explore the entire domain of loc and t with no restrictions on the number of triples in any window.

In a given window, LKC-privacy ensures that any subsequence of size up to L appears at least K times and the probability of inferring any victim's sensitive values is at most C. We formalize this model as follows.

Definition 4.5 (LKC-privacy)**.** Given a set of sensitive values Sen, a positive integer L, an anonymity threshold $K \geq 1$, and a confidence threshold $0 \leq C \leq 1$, a window $W_{x \to y}$ satisfies LKC-privacy iff for any subsequence q with $|q| \leq L$, $|G(q)| \geq K$ and $Conf(sen\_val|G(q)) \leq C$ for any $sen\_val \in Sen$. ■

Sen is defined by the data holder. In Example 4.1.1, sen_val_1 is the only sensitive value in the sensitive attribute. If a stream does not contain any sensitive values, then $Sen = \emptyset$. If a data holder wants to ignore the sensitive values altogether, then assigning $C = 100\%$ would let any subsequence q satisfy the condition $Conf(sen\_val|G(q)) \leq C$. Furthermore, should certain locations be deemed sensitive, the data holder can include such locations in Sen. This hallmark in our privacy model gives the data holder further flexibility in terms of privacy requirements.

Given an anonymous window, denoted by $\hat{W}_{x \to y}$, that satisfies LKC-privacy, the probabilities of successful identity and attribute linkage attacks are $\leq 1/K$ and $\leq C$, respectively. We note that the same degree of data utility is achieved from both $\hat{W}_{x \to y}$ and its static version, i.e., static trajectories.

4.2.3 Problem Statement

Our proposed method achieves anonymity through *suppression* by efficiently removing selected doublets from raw window $W_{x \to y}$ with the goal of preserving its utility. We perform *global suppression*: all instances of the selected doublet will be removed

from $W_{x \to y}$. For example, the anonymous window $\hat{W}_{2 \to 4}$ depicted in Table 4.2 is the result of suppressing all instances of doublets $b.2$ and $e.4$ from the raw window $W_{2 \to 4}$ in Table 4.1.

In the spirit of preserving data truthfulness, we aim at achieving anonymous data that is a subset of the raw data. Suppression does not require a pre-defined taxonomy tree, which is essential for performing *generalization* and may not be conveniently available in real-life scenarios, especially if the location universe changes dynamically.

Applying techniques based on local suppression is not feasible even though such techniques may cause less information loss than global suppression. We use global suppression because it takes advantage of the monotonicity property of Apriori. In contrast, this property does not hold for local suppression because the number of violations does not decrease monotonically with respect to local suppressions. For example, suppose a trajectory table contains the sequence $a.1 \to b.2$ with support $= 2$. Let $K = 2$ and $L = 2$. $a.1 \to b.2$ is not a violation. If $b.2$ was locally suppressed from one record only, then the resulting sequence $a.1 \to b.2$ becomes a new violation. As a result, applying local suppression requires an extra step to check for newly generated violations. The authors in [27] showed that such extra step is computationally costly. Therefore, to accommodate the transient, real-time nature of trajectory stream, we cannot afford local suppression.

Definition 4.6 (Anonymizing trajectory stream)**.** Given a trajectory stream S, a sliding window $W_{x \to y}$, and an LKC-privacy requirement, the problem is to efficiently publish a sequence of anonymous sliding windows over S such that suppressions are minimized. ■

k-anonymity and confidence bounding are special cases of LKC-privacy [42]. According to [95] and [121], achieving optimal k-anonymity and optimal confidence bounding is NP-hard. It follows that achieving optimal LKC-privacy, i.e., performing the least number of suppressions in any window, is also NP-hard. As a result,

anonymizing a sequence of windows over S with minimum number of suppressions is NP-hard. In the next section we propose a greedy algorithm that obtains a sub-optimal solution.

4.3 Anonymization Algorithm

In this section, we present Incremental Trajectory Stream Anonymizer (ITSA), our algorithm for incrementally anonymizing every window $W_{x \to y}$ on trajectory stream S by means of suppression. We identify all subsequences in $W_{x \to y}$ that *violate* a given LKC-privacy requirement. A window is anonymous when it contains no violations. We also present the dynamic tree structure of the window for efficient updates and data retrieval.

4.3.1 Incremental Identification of Violations

In order to publish an anonymous window, we need to make sure it does not contain any violation. We formally define a violation as follows.

Definition 4.7 (Violation)**.** Assume a given LKC-privacy requirement and a sliding window $W_{x \to y}$ over S. If any subsequence q in $W_{x \to y}$, with $1 \leq |q| \leq L$, has $1 \leq |G(q)| < K$ and/or $Conf(sen\_val|G(q)) > C$, then q is a violation. ■

A violation can be any possible combination of doublets in $W_{x \to y}$ that does not adhere to LKC-privacy. For example, $\langle b.2 \to d.4 \rangle$ in $W_{2 \to 4}$ (Table 4.1) is a violation, as Example 4.1.1 demonstrates. Eliminating all violations transforms $W_{x \to y}$ to an anonymous version $\hat{W}_{x \to y}$ that protects against privacy threats. Chapter 4.3.3 discusses how violations are efficiently suppressed.

If a window contains z distinct doublets, the total number of possible sequences to be checked is $2^z - 1$. Due to this exponential growth of candidate subsequences, we adopt the monotonicity property of Apriori [4] and only identify *critical violations*

instead of exhaustively finding all violations in a window. A critical violation is defined below.

Definition 4.8 (Critical violation)**.** A sequence v is a critical violation iff v is a violation and none of its subsequences is a violation. ■

If v has at least one subsequence v' that violates a given LKC-privacy requirement, then v is a violation but not a critical violation.

Example 4.3.1. Given window $W_{2\to4}$ in Table 4.1, let $L = 2$, $K = 2$, $C = 40\%$, and sen_val_1 be a sensitive value. $q_1 = \langle b.2 \to c.3\rangle$ is not a violation because $|G(q_1)| = 3 \geq 2$ and $Conf(sen\_val_1|G(q_1)) = 33\% \leq 40\%$. $q_2 = \langle b.2 \to c.3 \to d.4\rangle$ is a violation because, although $|G(q_2)| = 2 \geq 2$, $Conf(sen\_val_1|G(q_2)) = 50\% > 40\%$. On the other hand, q_2 is not a critical violation because one of its subsequences, $q_2' = \langle b.2 \to d.4\rangle$, is a violation. q_2' itself, however, is a critical violation because neither $b.2$ nor $d.4$ is a violation. ■

From Definition 4.8, we make the following observation about anonymizing a raw window.

Observation 4.1. Removing all critical violations from a raw window $W_{x\to y}$ transforms it to an anonymous version $\hat{W}_{x\to y}$, with respect to a given LKC-privacy requirement, that contains no violations. ■

Proof. Let v_1 be a critical violation due to $|G(v_1)| < K$. Then any supersequence v_1'' of v_1 is a violation because $|G(v_1'')| \leq |G(v_1)| < K$. However, any subsequence v_1' of v_1 is not a violation because $|G(v_1')| \geq |G(v_1)|$. Therefore, if v_1 satisfies LKC-privacy then v_1 also satisfies $L'KC$-privacy, for $L' < L$.

Let v_2 be a critical violation due to $Conf(sen\_val|G(v_2)) > C$. A supersequence v_2'' of v_2 may or may not be a violation because there exists no relation between $Conf(sen\_val|G(v_2))$ and $Conf(sen\_val|G(v_2''))$. Therefore, according to Definitions 4.5 and 4.8, v_2'' will not be in any $Cand_i$. □

We iteratively generate the set of all i-size candidate subsequences, $Cand_i$, by self-joining non-violating subsequences in $Cand_{i-1}$. Every subsequence $q \in Cand_i$ is checked against the given privacy requirement. If q is a (critical) violation, it is removed from $Cand_i$. To mitigate information loss due to suppression, Chapter 4.3.3 shows that we do not actually need to remove all occurrences of a critical violation v from $W_{x \to y}$; rather, removing specific doublets in v is sufficient.

We identify below certain properties in a trajectory stream S and integrate them in building an efficient algorithm for incrementally anonymizing a sliding window over S. Recall the subsequent window of $W_{x \to y}$ is denoted by $W_{x^+ \to y^+}$ (Definition 4.4).

Property 4.1. When anonymous window $\hat{W}_{x \to y}$ slides, $\hat{W}_{x \to y} - O$ incurs no violations. ■

Removing the set of outdated doublets O from anonymous window $\hat{W}_{x \to y}$ does not create violations. This is because: (a) all doublets in O are globally suppressed from $\hat{W}_{x \to y}$, and (b) according to Definition 4.5, all subsequences of size up to L satisfy the privacy requirement.

Property 4.2. If subsequence q in anonymous window $\hat{W}_{x \to y}$ satisfies LKC-privacy, then q also satisfies LKC-privacy in any subsequent window that contains q. ■

Let q be a non-violation in anonymous window $\hat{W}_{x \to y}$. If subsequent raw window $W_{x^+ \to y^+}$ contains q, then q is still a non-violation.

Property 4.3. When anonymous window $\hat{W}_{x \to y}$ slides, $\bigcup E$ may create new violations. ■

Let q be a non-violation in anonymous window $\hat{W}_{x \to y}$. Adding the set of new doublets E to the subsequent raw window $W_{x^+ \to y^+}$ creates new combinations of doublets. Consequently, $q \cup \eta$, where η is a subsequence from E, may or may not be a violations.

Algorithm 4.1: Incremental Trajectory Stream Anonymizer (ITSA)

Input: Anonymous window $\hat{W}_{x \to y}$. Trajectory stream S
Input: Thresholds L, K, C, and sensitive values Sen
Input: Window size N and $step\_size$
Output: New anonymous window $\hat{W}_{x^+ \to y^+}$

1: **while** S exists and $L \leq N$ **do**
2: slide $\hat{W}_{x \to y}$ over S to get raw $W_{x^+ \to y^+}$;
3: obtain E;
/*Phase 1*/
4: $\hat{E}$ = AnonymizeNew(L, K, C, $step\_size$, E);
5: $Cand_1$ = all unique doublets in $W_{x^+ \to y^+}$. $Cand_1 \cap \hat{E} = \emptyset$;
6: $Cand_2 = Cand_1 \bowtie \hat{E}$;
7: let $V = \emptyset$;
8: let $i = 2$;
/*Phase 2*/
9: **repeat**
10: **for all** $q \in Cand_i$ **do**
11: **if** $\exists v \in V$ s.t. $v \subseteq q$ **then**
12: remove q from $Cand_i$ and add q to V;
13: **else**
14: **if** $|G(q)| < K$ or $Conf(sen\_val|G(q)) > C, \forall sen\_val \in Sen$ **then**
15: remove q from $Cand_i$ and add q to V;
16: **end if**
17: **end if**
18: **end for**
19: **if** $i++ \leq L$ **then**
20: $Cand_i = Cand_{i-1} \bowtie Cand_{i-1}$;
21: **end if**
22: **until** $i > L$ or $Cand_i = \emptyset$
/*Phase 3*/
23: Win = findWinners(V);
24: **for all** $w \in Win$ **do**
25: remove all instances of winner doublet w from $W_{x^+ \to y^+}$;
26: **end for**
27: **Publish** $\hat{W}_{x^+ \to y^+}$;
28: **end while**

Algorithm 4.1 (ITSA). This algorithm runs every time the window slides over a trajectory stream S. Suppose that anonymous window $\hat{W}_{x \to y}$ of size N has just been published. Line 2 slides the window by *step_size* timestamps. Outdated doublets O drop out and new doublets E arrive. The updated window, $W_{x^+ \to y^+}$, contains raw data. Phase 1 anonymizes E, Phase 2 obtains all critical violations V, and Phase 3 removes V from the raw window. Finally, anonymous $\hat{W}_{x^+ \to y^+}$ is published.

Phase 1. As a preprocessing step, we first anonymize E. Any subsequence η from E is a special case to which Property 4.3 applies. Consequently, any subsequence η with $|\eta| \leq L$ is checked. The maximum length of any η is equal to *step_size*, which is relatively small compared to window size N. Algorithm 4.2 applies LKC-privacy on raw E. Critical violations found in E are removed from $W_{x^+ \to y^+}$. Line 6 creates the 2-size candidate set $Cand_2$ by self-joining $Cand_1$ and $\hat{E}$. This process is demonstrated below.

Phase 2. We identify all critical violations V in $W_{x^+ \to y^+}$. Thanks to Properties 4.2 and 4.3, we check only subsequences that contain at least one doublet from E. This phase iteratively generates $Cand_i$ (Line 20) to check for critical violations. The iteration is terminated when i exceeds L or $Cand_i$ can not be generated. Two subsequences, $q_y = \langle (loc_1^y.t_1^y) \to \ldots \to (loc_{i-1}^y.t_{i-1}^y) \rangle$ and $q_z = \langle (loc_1^z.t_1^z) \to \ldots \to (loc_{i-1}^z.t_{i-1}^z) \rangle$, can be self-joined only if all doublets except the last (the one having t_{i-1}) are identical in both subsequences, and $t_{i-1}^y < t_{i-1}^z$. The resulting sequence is $\langle (loc_1^y.t_1^y) \to \ldots \to (loc_{i-1}^y.t_{i-1}^y) \to (loc_{i-1}^z.t_{i-1}^z) \rangle$. Lines 10-12 ensure that a candidate subsequence q is not a supersequence of a violation in V.

Phase 3. We remove all critical violations from $W_{x^+ \to y^+}$. Line 23 calls Algorithm 4.3 in order to suppress only selected doublets in V from $W_{x^+ \to y^+}$. Finding these doublets, referred to as *winner doublets* Win, for suppression is motivated by the goal of incurring less impact on the data utility. This process is detailed in

Algorithm 4.2: *AnonymizeNew*

Input: New doublets E, and $step\_size$
Input: Thresholds L, K, C, and Sensitive values Sen
Output: Anonymous $\hat{E}$

1: let $V = \emptyset$;
2: **for** ($i = 1$, $i \leq min(L, step\_size)$, $++i$) **do**
3: Generate every possible i-size sequence η from E;
4: Scan E once to find $|G(\eta)|$ and $Conf(sen\_val|G(\eta))$ for any $sen\_val \in Sen$;
5: **if** $|G(\eta)| < K$ or $Conf(sen\_val|G(\eta)) > C$ for any $sen\_val \in Sen$ **then**
6: Add η to V;
7: **end if**
8: **end for**
9: **if** $V = \emptyset$ **then**
10: **return** E as anonymous $\hat{E}$;
11: **else**
12: Win = findWinners(V);
13: **for all** $w \in Win$ **do**
14: Suppress w from E;
15: **end for**
16: **return** $\hat{E}$ as anonymous version of E;
17: **end if**

Chapter 4.3.3. Line 27 publishes the anonymous window $\hat{W}_{x^+ \rightarrow y^+}$ after all winner doublets have been removed.

Example 4.3.2. We continue from Example 4.3.1. As $\hat{W}_{1\rightarrow 3}$ slides, $E = \{\langle 1, d, 4\rangle, \langle 2, d, 4\rangle, \langle 3, d, 4\rangle, \langle 7, d, 4\rangle, \langle 8, e, 4\rangle\}$. First, Algorithm 4.2 in Phase 1 determines that $e.4$ is a violation and, thus, suppresses $e.4$ from E resulting in $\hat{E}$. Second, Phase 2 generates and tests every candidate subsequence q containing existing and new doublets in $W_{2\rightarrow 4}$. $Cand_2 = \{\langle b.2 \rightarrow d.4\rangle, \langle c.3 \rightarrow d.4\rangle, \langle f.2 \rightarrow d.4\rangle\}$. $q_1 = \langle b.2 \rightarrow d.4\rangle$ is a critical violation because $Conf(sen\_val_1|G(q_1)) = 50\% > 40\%$. To demonstrate self-joining, let $L = 3$. Then, $Cand_3 = \{\langle f.2 \rightarrow c.3 \rightarrow d.4\rangle\}$. Note that $Cand_3$ does not include $\langle b.2 \rightarrow c.3 \rightarrow d.4\rangle$ because it is a supersequence of q_1. ■

Algorithm 4.3: *findWinners*

Input: Critical violations V
Output: Winner doublets Win

```
1: Initialize score table;
2: Let Win = ∅;
3: repeat
4:     Choose winner doublet w with the highest Score;
5:     Add w to Win;
6:     Remove w from score table and update it accordingly;
7: until Score table is empty
8: return Win;
```

4.3.2 Sliding Window as a Tree

When the sliding window moves, a relatively small fraction of the data is added/ dropped while the remaining larger portion does not change (overlapping data). Our proposed method efficiently handles this transition. Moreover, our method allows the sliding window to add/remove trajectories of joining/leaving individuals. ITSA entails adding, removing, and searching sequences of doublets. We explain below how our method facilitates these operations on the window.

We use a *trie* structure to implementing sliding window. A trie is a tree data structure where each node is a prefix to all its descendants. The trie is created once (first window) then is dynamically updated upon every window slide. Our tree structure is reminiscent of the FP-tree structure introduced in FP-growth, a method for mining frequent patterns [59].

Definition 4.9 (Trie)**.** A trie, $\mathbb{P} = (Nodes, Edges, Root)$, of trajectories in $W_{x \to y}$ consists of a collection of nodes, edges, and a Root node. Every node $n \in Nodes$ contains a doublet b from $W_{x \to y}$ and a prefix count, *count*. The *count* of node n stores the number of distinct trajectories containing the prefix subsequence in the unique $Root - to - node$ path. The sequence of doublets on a $Root - to - leaf$ path constitutes a full trajectory. The Root node contains only a *count*, which is the total number of trajectories. ■

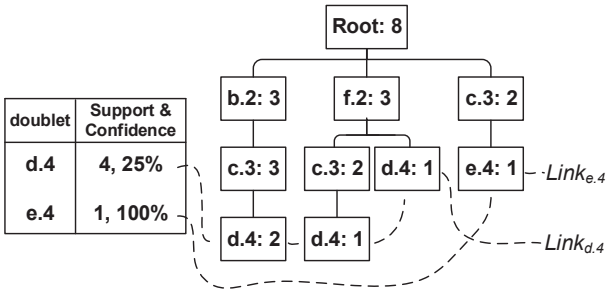


Figure 4.2: Raw window $W_{2\to4}$ in Table 4.1 structured as a trie

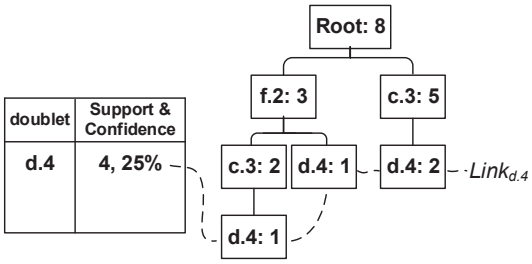


Figure 4.3: Anonymous window $\hat{W}_{2\to4}$ in Table 4.2 structured as a trie

The trie structure has a significant impact on storage space by allowing a concise view over trajectories. Let us examine raw window $W_{2\to4}$ in Table 4.1. $W_{2\to4}$ contains three instances of the sequence $\langle b.2 \to c.3\rangle$. Figure 4.2 is a trie representation of $W_{2\to4}$. We notice that only two nodes are used to represent all the three instances of this sequence.

When the window slides, nodes containing new doublets from $\hat{E}$ are added as leaves. Moreover, nodes on the higher levels are deleted if they contain doublets O. Figure 4.3 shows anonymous window $\hat{W}_{2\to4}$, where nodes containing $a.1 \in O$ and $b.2, e.4 \in Win$ are removed.

When generating candidate subsequences from trie $\mathbb{P}$, a subsequence $q \in Cand_i$ may appear in several branches in $\mathbb{P}$. This is challenging when counting the total number of trajectories containing q (i.e., the *support* of q). A naive way is exhaustively search $\mathbb{P}$. Instead, we use a virtual line, called $Link$, to connect all nodes containing the same doublet in $\mathbb{P}$. Figures 4.2 and 4.3 show $Link_{e.4}$ and $Link_{d.4}$.

Definition 4.10 (*Link*)**.** Given a trie $\mathbb{P}$, $Link_b$ is a sequence of positions that belong to all the nodes containing doublet b in $\mathbb{P}$. ■

Finding the support and confidence of a doublet b is achieved by traversing $Link_b$. Simply adding up the *counts* of all the nodes on $Link_b$ yields the support of b, i.e., $|G(b)|$. To calculate $Conf(sen\_val|G(b))$, we use a separate *sensitive count* for every unique sensitive value in Sen. Summing up the sensitive counts pertaining to sensitive value sen_val yields $|G(b \bigcup sen\_val)|$, which is used along with the previously found support to calculate the confidence.

Example 4.3.3. Figure 4.2 shows the trie $\mathbb{P}$ of the raw window $W_{2\rightarrow4}$. New nodes of $d.4, e.4 \in E$ are added as leaves. The node of $d.4$ in the sequence $q_1 = \langle b.2 \rightarrow c.3 \rightarrow d.4\rangle$ has $count = 2$. This indicates that two instances of $d.4$ preceded by $\langle b.2 \rightarrow c.3\rangle$ appear in two trajectories in $W_{2\rightarrow4}$. These two trajectories, as shown in Table 4.1, are associated with $sen\_val_1 \in Sen$ and sen_val_3. Therefore, the node of $d.4$ has sensitive count = 1. Hence, $Conf(sen\_val_1|G(q_1)) = 1/2 = 50\%$. ■

From the above discussion, we can find the support of a given sequence $q = \langle b_1 \rightarrow b_2 \rightarrow b_3 \rightarrow \ldots \rightarrow b_{|q|}\rangle$ in a given trie $\mathbb{P}$ as follows. Since every node in $\mathbb{P}$ is a prefix to all its descendants, it follows that $count_{b_i} \leq count_{b_j}$, where $j < i$. This finding implies that $count_q = count_{b_{|q|}}$. To find the support of q, we need to trace $Link_{b_{|q|}}$ to check for other instance of q in other branches of $\mathbb{P}$. The support of $q = |G(q)| = \sum count_{b_{|q|}}$, given q exists in these branches. Calculating the confidence follows the same reasoning.

Example 4.3.4. Let us review q_1 in Example 4.3.3. The leaf node of $d.4$ has $count = 2$. Consequently, any subsequence of q_1 containing $d.4$ will have its $count = 2$. Let $q'_1 = \langle c.3 \rightarrow d.4\rangle \subseteq q_1$. q'_1 has two instances in two separate branches in $\mathbb{P}$. The support of $q'_1 = \sum count_{d.4} = 2 + 1 = 3$. ■

As we pointed out earlier, the trie is created once (in the first window). Only at this one point does Algorithm 4.2 run on an entire window since all doublets in

the first window are new, where $step\_size = N$. Afterwards, the trie is adjusted dynamically as new doublets arrive and outdated doublets drop out. Hence, only new doublets E are read from S.

4.3.3 Suppression

After identifying all critical violations V in raw window $W_{x \to y}$ (Phase 2 in Algorithm 4.1), we need to remove them from $W_{x \to y}$. A naive approach is to remove every instance of a critical violation $v \in V$. Although this approach will result in an anonymous window that satisfies LKC-privacy, the incurred data loss (referred to it as *distortion*) would be significantly high. Instead, we propose a greedy procedure that selects certain *winner* doublets from V for suppression.

When a doublet w is chosen for suppression, w is globally suppressed. That is, all instances of w are removed from $W_{x \to y}$. Global suppression does not increase the maximum probability of a successful attack (identity and attribute linkages) because any proper subsequence in $\hat{W}_{x \to y}$ still satisfies the imposed LKC-privacy requirement.

$$Score(d) = \frac{PrivGain(d)}{InfoLoss(d)} \tag{4.1}$$

Removing a violating sequence v implies that: (a) we are gaining privacy by removing an attack channel, and (b) we are inflicting data distortion due to permanently removing doublets from the original trajectories in $W_{x \to y}$. Hence, we adopt a greedy function that attempts at removing all violating sequences in V by suppressing selected winner doublets, Win. We apply Equation 4.1 to every distinct doublet b in V. $PrivGain(b)$ is the number of critical violations containing b, and $InfoLoss(b)$ is the support of b in the current raw window. The doublet with the highest $Score$ is labeled a winner doublet w and is added to Win.

Algorithm 4.3 finds the winner doublets Win from the set of all critical violations V produced by Phase 2 in Algorithm 4.1. Line 1 creates the score table and fills in the values of $PrivGain(b)$ and $InfoLoss(b)$ pertaining to every unique doublet b in V. Lines 3-7 iterate the score table to find a winner doublet. The purpose is to find a doublet that its removal would result in the removal of a maximum number of critical violations yet causing least distortion. Therefore, Algorithm 4.3 selects the doublet w that has the highest $Score$. Then, w is added to the set of winner doublets Win. After that, Algorithm 4.3 updates the score table by removing w and adjusting the value of $PrivGain(b)$ of every doublet b that coexists with w in the same critical violation. If b no longer exists in V, b's entry is removed from the score table. Line 8 returns Win containing all winner doublets to be suppressed from raw $W_{x \to y}$.

4.3.4 Complexity Analysis of ITSA

The algorithm starts by sliding the window over stream S (Line 2). This process requires reading the set of new doublets E from S and inserting the new subsequences into the window. The former step is proportional to the total number of individuals, $|P|$. The latter step costs $O(N.|\mathcal{L}|.|\eta|)$, where $\mathcal{L}$ is the location universe, η is a new subsequence from E, and $|\eta| \leq step\_size$.

We explain the complexity of generating $Cand_i$ (Phase 2). Let U be the set of all distinct doublets in a given window, and let $2 \leq i \leq L$. $Cand_i$ is a set of i-size *combinations* from U. The number of i-size combinations from a set of $|U|$ doublets $\binom{|U|}{i} = \frac{|U|(|U|-1)...(|U|-i+1)}{i(i-1)...1} = O(|U|^i)$. The worst-case scenario would be for $i = L$, i.e., generating L-size sequences of doublets from U requires $O(|U|^L)$, where $L \leq N$. However, this exponential behavior is limited due to the following reasons. First, $Cand_i$ is achieved by self-joining the sequences in $Cand_{i-1}$ having the same prefix. It is less likely, however, to find long sequences sharing the same prefix. Second, a sequence $q \in Cand_i$ is removed from $Cand_i$ if q is a supersequence of a previous

critical violation. For these two reasons, increasing i does not have a significant effect on the size of $Cand_i$.

Let $\eta \in E'$ be a subsequence that exists in $q \in Cand_i$, and let $|Link_\eta|$ denote the number of branches containing the full sequence $q \in Cand_i$ in the current trie. Phase 2 is bounded by $O(|U|^L.|Link_\eta|)$.

Phase 1 of Algorithm 4.2 generates $Cand_i$ in a similar fashion as described above. However, for any sequence q in this case, $|q| \leq min(L, step\_size) \leq N$.

To search for a single winner doublet w (Phase 2) in a trie $\mathbb{P}$, we only need to follow $Link_w$. When updating $\mathbb{P}$ after deleting a single w node, the child nodes of w go up by one level. Nodes containing the same doublet are merged. Any level e contains at most $(N - e + 1)|\mathcal{L}|$ nodes.

In summary, the most costly operation in ITSA is candidate set generation, described in Phase 2. The complexity of ITSA is dominated by attacker's knowledge L, i.e., $O(|U|^L)$, where $L \leq N$. Practically, however, Phase 2 is most likely to terminate in early iterations due the aforementioned reasons.

4.3.5 Discussion

We discuss two types of privacy attacks on the published data. These attacks differ from the privacy attacks we introduced in Chapter 4.1. Furthermore, we discuss the situation where adversary's knowledge covers more than one published window.

Attacks. The first attack is called the *minimality attack* [126]. It stems from knowing that a certain anonymization method does not anonymize data beyond a *minimum* point at which the data satisfy a given privacy requirement. Minimality provides less data distortion, allowing for better data utility. This attack is based on the ability of an adversary to reconstruct raw data from its anonymous version. Data reconstruction is made possible when there exists a public dataset with matching quasi-identifier attributes. From the reconstructed data, the adversary would

be able to narrow down to the portions of the data that contain potential violations. However, our method produces anonymous trajectories that are subsets of their pertinent raw versions by a sequence of suppressions. By looking at an anonymous trajectory, it is impossible to conclude if suppression took place and on which doublets. Also, we assume raw data only show doublets when a traveler changes locations. For example, Record 8 in Table 4.2 contains only $c.3$. An adversary would not be able to tell if two doublets at timestamps 2 and 4 were suppressed or the traveler stayed at location c at timestamp 4.

The second attack is *correlation attack*. It takes place when a doublet is widely known by data recipients to be highly associated with a sequence of other doublets. As an example, most drivers in the *downtown* area taking the *highway* cross over the *bridge* to get to the *mall*. If the sequence $\langle downtown.1 \rightarrow highway.2 \rightarrow mall.4 \rangle$ exists in the published data, then an adversary will be able to deduce the missing doublet of the target victim, $bridge.3$. We point out that if such association exists (i.e., high confidence), it is unlikely that ITSA would suppress the recurring doublet since it would have a high support. Moreover, this work does not treat locations as sensitive information. If locations are sensitive, some methods already exist that incorporate this assumption in their solution [117]. Alternatively, sensitive locations can be added to the set of sensitive values with no effect on the performance of ITSA.

Adversary's Knowledge. The sliding window contains recent whereabouts of the moving individuals, since recent movements are most useful for data mining tasks. However, if adversary's knowledge spans multiple windows, the data holder needs to adjust the window size so that a wider range of trajectories is anonymized.

This solution, though simple and practical, suffers from two limitations. First, it is not always trivial to deduce the length of the adversary's knowledge, L. Second, from Chapter 4.3.4, we can see that larger L incurs higher complexity. To circumvent these drawbacks, in a future work we will consider anonymizing sequential windows so that recent windows are not published independently. In other words, ITSA will

still publish one anonymous window at a time, but recent windows will cover wider adversary's knowledge. One way to accomplish this goal is to maintain a buffer containing count statistics from a certain number of recently published windows. The algorithm would then anonymize the current window based on the information available about the recently published anonymous windows.

4.4 Performance Analysis

We implemented our method in C++. All experiments run on an Intel Core i5 CPU with 2.4GHz and 4GB of RAM. We evaluate the performance of our method in terms of data utility, efficiency, and scalability.

We carry out performance evaluation using three datasets. The first one is a simulated dataset called *MetroData*[1]. The second one is a real-life dataset called *MSNBC*, which is publicly available at the UCI machine learning repository [10]. The third dataset is generated using Brinkhoff's network-based synthetic data generator [19] and is called *Oldenburg*. *MetroData* is a simulation of the traffic routes of a large group of passengers using the public transit metro system in Montreal, Canada. Routes are generated based on the information and statistics provided in an annual report published by www.metrodemontreal.com. The generator that we built takes into consideration the actual metro map and the passenger flow rate of each station. According to the published statistics, a passenger passes through 8 stations on average.

MetroData contains trajectories generated for $100,000$ passengers who use any of the metro's 65 stations over a 60-minute time period. Therefore, *MetroData* includes 100,000 records, each record belongs to a unique passenger. The total dimensionality of the dataset is $65 \times 60 = 3900$ dimensions. We also assume the existence of a sensitive attribute, namely *social_status*. We note that a sensitive

[1] http://dmas.lab.mcgill.ca/fung/pub/MetroDataSet.txt

attribute could be any attribute chosen or imposed by the data holder; we choose *social_status* as an example. *social_status* has five domain values, we choose *On-Welfare* to be sensitive. In this case, the set of sensitive values $Sen = \{On - Welfare\}$. All records are evenly assigned one value from the sensitive attribute.

The second dataset, *MSNBC*, is a real-life web log containing visited web pages by nearly 1 million users, where every record belongs to a unique user. The dataset contains 17 categories of web pages: News, Tech, Health, etc. Despite the fact that this dataset contains no physical locations, it shares the same property of high dimensionality with a typical trajectory dataset. Therefore, categories of web pages are treated as unique locations visited by users at non-decreasing timestamps. We also impose a sensitive attribute on the original data. The sensitive attribute contains 10 domain values, each record is randomly assigned one value. We choose two domain values to be sensitive.

The last dataset, *Oldenburg*, is generated using Brinkhoff's network-based traffic generator. *Oldenburg* contains 100,000 trajectories of objects moving throughout the city of Oldenburg (Germany) over the course of 24 hours. The representation of the generated trajectories has been modified to adhere to Definition 4.1. Hence, as a preprocessing step, the road-network map of the city of Oldenburg was discretized into $10 \times 10 = 100$ regions, and all X-Y coordinates of the generated trajectories were replaced with their pertinent regions. In addition, a sensitive value is randomly assigned to every trajectory in a similar setting as in *MetroData*.

To simulate a streaming environment, the window slides over trajectories reading only doublets that fit within the window scope. Algorithm 4.1 runs only on the data available in a given window. That is, any future or outdated doublets are unknown to the algorithm. Without loss of generality, we set $step\_size = 1$ in all experiments.

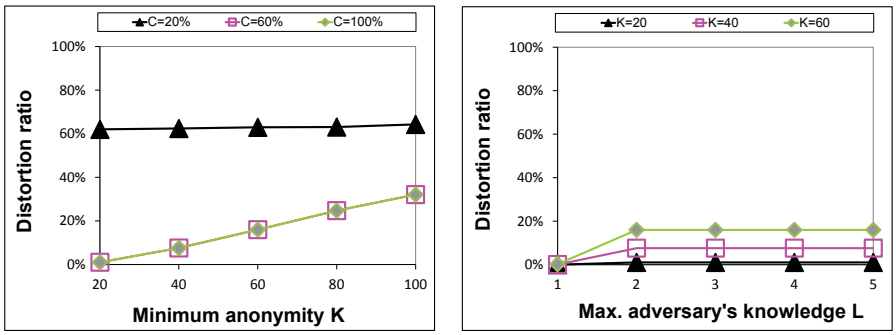
We compare our method ITSA with two other methods, namely RFIDAnonymizer [42] and Never Walk Alone (NWA) [1]. RFIDAnonymizer was previously proposed for anonymizing a static RFID dataset. To apply this static anonymization method on trajectory stream, we apply RFIDAnonymizer on every window independently. We compare our method with RFIDAnonymizer only in terms of efficiency and scalability, but not in terms of data utility, because both methods apply exactly the same sequence of suppression operations, yielding the same result. The contribution of ITSA over RFIDAnonymizer is on efficiency and scalability. We also compare our method with another method called NWA that was proposed by Abul et al. [1] to anonymize a static trajectory dataset. We carry out performance analysis with NWA in terms of data utility and efficiency. In addition, we compare the impact of applying different privacy models, namely LKC-privacy and k-anonymity, on the data utility.

Unless otherwise specified, a value on the distortion ratio and runtime reported in any of the figures in this section reflects the average value of all the windows sliding over the same dataset in one run.

4.4.1 *Metro* Dataset

First, we study the impact of the different parameters on data utility. Then, we generate much larger datasets to test the scalability of our method.

Figure 4.4 (a). We fix N to 7 and the adversary's background knowledge L to 4. We vary the minimum anonymity threshold K from 20 to 100 and the maximum inference confidence C from 20% to 100%. We see that increasing K implies higher distortion ratio. This is because more trajectories must share any subsequence q with $|q| \leq L$, and it is unlikely that a large number of passengers share longer journey routes. Therefore, more suppressions are needed to achieve anonymity. We also notice that the distortion ratio is highest at $C = 20\%$. We attribute this odd jump to the random distribution of the sensitive values in each



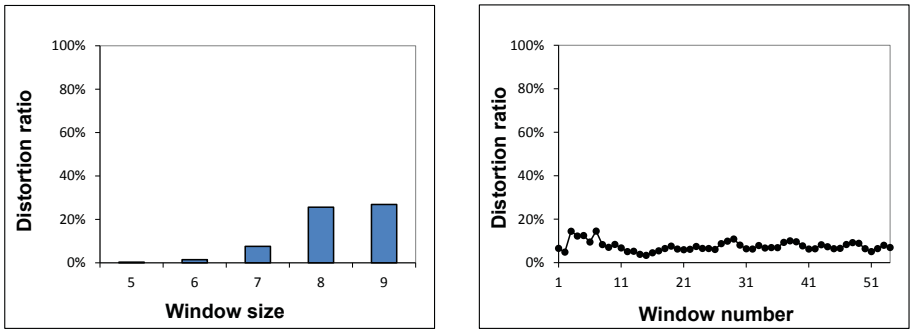
(a) Dist. Ratio vs. K and C (b) Dist. Ratio vs. L and K

Figure 4.4: *MetroData*: the impact of L, K, C ($N = 7$)

record, 1 out of $5 = 20\% = C$. As a result, more doublets are suppressed to satisfy the inference confidence requirement.

Figure 4.4 (b). We fix N to 7 and C to 60%. We vary L from 1 to 5 and K from 20 to 60. For $L \geq 2$, the distortion ratio experiences a steady behavior because the values of K are relatively small compared to the total number of records in the dataset. We note that in real-life datasets, the distortion ratio is expected to be proportional to L. This is because in order to satisfy LKC-privacy, more trajectories must share longer subsequences; otherwise, further suppressions are required. Figure 4.4 (b) also affirms that distortion ratio increases when K increases.

Figure 4.5 (a). We evaluate the impact of N. We fix L to 4, K to 40, and C to 60%. We vary N from 5 to 9. Figure 4.5 (a) insinuates that larger window sizes are likely to cause higher distortions. This is because a larger window size produces a much larger $Cand_i$, resulting in increasing the number of potential critical violations. Generally speaking, as the window slides, new data may or may not introduce new critical violations (Property 4.3), and thus there is no formal approach for systematically predicting an association between window size and distortion. This reasoning is reflected in Figure 4.5 (a) when $5 \leq N \leq 7$ where the distortion ratio increases slowly. However, the distortion ratio experiences a sudden jump at $N = 8$, but has a marginal increase at $N = 9$.



(a) Dist. Ratio vs. N (b) Dist. Ratio in Windows ($N = 7$)

Figure 4.5: *MetroData*: sliding window ($L = 4$, $K = 40$, $C = 60\%$)

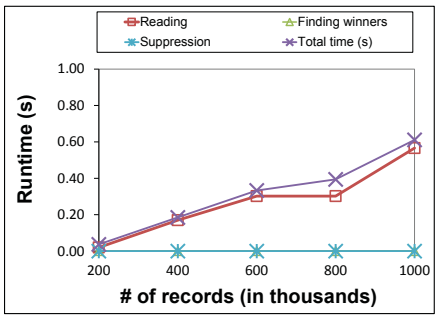
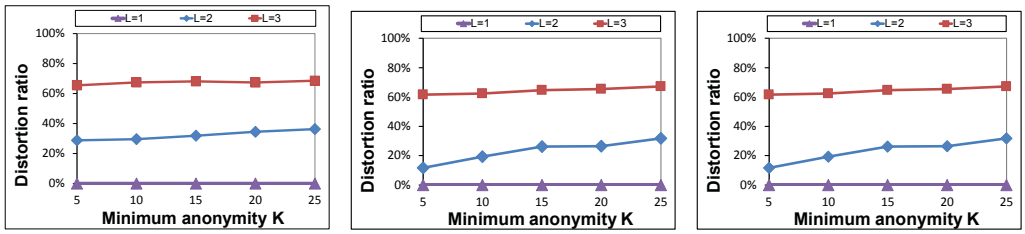


Figure 4.6: *MetroData*: scalability ($L = 4$, $K = 120$, $C = 60\%$, $N = 7$)

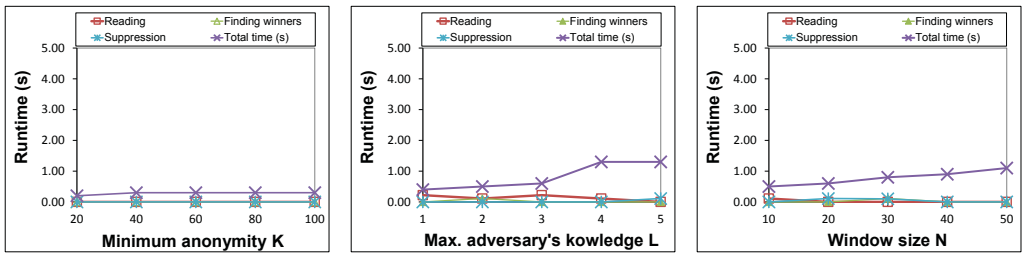
Figure 4.5 (b). All previous figures were depicting the distortion ratio on average for all windows. Figure 4.5 (a) presents the distortion ratio of every window on *MetroData*. We set $L = 4$, $K = 40$, $C = 60\%$, and $N = 7$. We see that the distortion ratio exhibits mild fluctuation as the window slides.

Efficiency and Scalability. In all the experiments thus far, the runtime of anonymizing a single window before it slides is less than 1 second. We now evaluate the efficiency and scalability of our method in terms of handling datasets with a huge number of records. Our method puts no restrictions on the amount of data to be processed in a sliding window. The maximum number of doublet instances in any window is equal to window size $N\times$ the number of records [83] [123] [144]. We use our generator to produce datasets with size ranging from 200 thousand to



(a) $C = 20\%$ (b) $C = 60\%$ (c) $C = 100\%$

Figure 4.7: *MSNBC*: distortion ratio vs. L, K, C



(a) $C = 60\%, L = 2, N = 5$ (b) $K = 20, C = 60\%, N = 5$ (c) $K = 20, C = 60\%, L = 2$

Figure 4.8: *MSNBC*: runtime vs. K, L, N

1 million records. We set $L = 4$, $K = 120$, $C = 60\%$, and $N = 7$. Figure 4.6 shows that a sliding window takes less than 1 second to anonymize 1 million records.

4.4.2 *MSNBC* Dataset

For our second dataset, we perform more in-depth analysis and observe the impact of the different parameters on distortion ratio and runtime. All experiments consider the first 10 windows sliding over the entire dataset.

Figure 4.7. We evaluate the impact of L, K, and C on distortion ratio. Experiments are carried out for window size $N = 5$, anonymity threshold $5 \leq K \leq 25$, maximum adversary's background knowledge $1 \leq L \leq 3$, and confidence threshold $C = 20\%, 60\%, 100\%$. Overall, distortion ratio increases as K increases. At $L = 1$ (sequences of one doublet), the distortion ratio stays below 1% mainly because *MSNBC* contains a very large number of records, and thus unique doublets are bound to appear more frequently.

Figure 4.7 also shows that distortion ratio is proportional to L. Higher values of L imply that longer sequences must exist more frequently. Since it is unlikely that too many users would visit the same sequence of web pages, many doublets have to be suppressed to satisfy the given privacy requirement. Setting $L = 3$ produces a high distortion of $\geq 60\%$ in Figures 4.7 (a)-(c).

Lastly, Figure 4.7 (a), where $C = 20\%$, reports a higher distortion ratio at $L = 2$ (30% to 40%) than Figures 4.7 (b) and (c) (10% to 30%). This is justified by how sensitive values are assigned to each record. Since the sensitive attribute contains 10 domain values, 2 of which are sensitive, each record has a probability of 20% to be assigned a sensitive value, which is equal to the confidence threshold. Therefore, more suppressions are performed. This observation is reflected in Figures 4.7 (b) and (c) that report lower distortions.

Figure 4.8. We study the effect of K, L, and N on the runtime of our method. For every graph, we vary one parameter and fix the others (values are in caption). In Figure 4.8 (a), for $20 \leq K \leq 100$, runtime stays below 0.5 seconds. Moreover, we note that runtime is insensitive to the change of K. This is because the number of critical violations does not grow significantly as K increases, hence the time to find winner doublets is not generally affected. This observation is depicted in Figure 4.7 (b) at $L = 2$ where the distortion ratio increases slowly as K goes higher. The same reasoning can be applied to the inference confidence, C.

Figure 4.8 (b) depicts the impact of L on runtime. The maximum value that can be assigned to L is the window size N. Therefore, we vary L between 1 and N. In general, the runtime increases as L increases. For $1 \leq L \leq 5$, $0.4 \leq runtime \leq 1.3$ seconds. As L increases, more sequences are generated to be checked for potential critical violations. This fact is also reflected in Figure 4.8 (c) where $10 \leq N \leq 50$. Larger values of N imply more unique sequences in any candidate set $Cand_i$, hence more processing time.

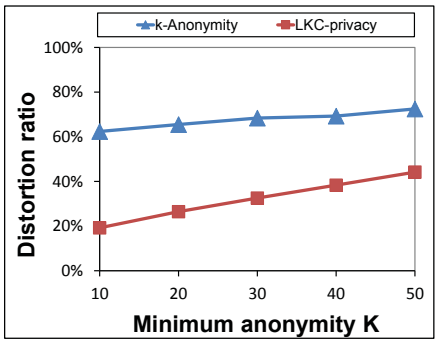


Figure 4.9: k-Anonymity vs. LKC-privacy ($C = 60\%, L = 2, N = 5$)

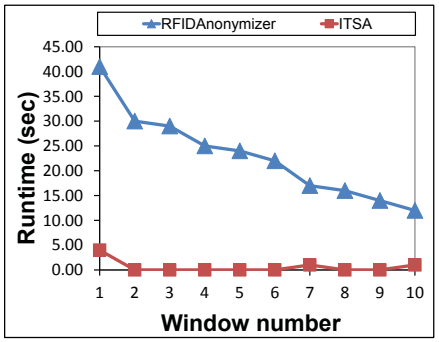


Figure 4.10: RFIDAnonymizer vs. ITSA ($K = 20, C = 60\%, L = 2, N = 10$)

Figure 4.9. We compare the distortion ratios caused by applying the LKC-privacy model and the traditional k-anonymity model, respectively, in our ITSA method. By setting $L = N$ and $C = 100\%$, LKC-privacy turns into k-anonymity. For LKC-privacy, we set $C = 60\%, L = 2, N = 5$ and $10 \leq K \leq 50$. Results in Figure 4.9 show that k-anonymity maintains a minimum distortion ratio of 60%. Applying LKC-privacy significantly lowers the distortion ratio over the different values of K. This can be explained by the fact that k-anonymity requires every record to be shared in its entirety by at least $k - 1$ other records. This strict requirement is addressed in LKC-privacy by manipulating the parameter L.

Figure 4.10. We measure the efficiency and scalability of our method in terms of runtime (in seconds), and compare the results with those achieved by using RFIDAnonymizer [42]. For this experiment, we measure the runtime of every individual window when ITSA and RFIDAnonymizer are applied, independently. Figure 4.10

depicts the runtimes of 10 windows. It is evident that ITSA performs significantly better than RFIDAnonymizer by maintaining an overall runtime of $\leq$ 1 second, thanks to the properties that we identified in trajectory streams (Chapter 4.3.1) and the compact and dynamic tree structure for representing trajectories (Chapter 4.3.2). Moreover, when the window slides, ITSA reads only the new data in the stream, while RFIDAnonymizer reads every window in its entirety.

4.4.3 *Oldenburg* Dataset

In this set of experiments, we compare the performance of our method ITSA with another method called NWA that was proposed by Abul et al. [1] to anonymize static trajectory data. We carry out performance analysis of both methods in terms of data utility and efficiency using the *Oldenburg* dataset.

NWA applies anonymization through *space translation*, by which trajectory points are either suppressed or dragged in space until every trajectory coexists with at least $k-1$ other trajectories. A translated point is assigned a penalty equal to the translation distance. If a point is suppressed, it is assigned a penalty equal to the *max point translation* (a constant value corresponding to the maximal translation distance in the entire experiment). This penalty metric is called *Information Distortion*, and is used in our experiments to evaluate the utility of the output anonymous dataset. Higher *Information Distortion* implies less data utility.

Our method ITSA applies anonymization through a sequence of suppressions. To compute *Information Distortion*, every suppressed point is assigned a penalty equal to the *max point translation* obtained from applying NWA on the same dataset.

Figure 4.11. We compare ITSA with NWA in terms of data utility by measuring *Information Distortion*. For ITSA, we set $C = 60\%, L = 4$, and $N = 5$. As for NWA, we mostly use the default parameters values; specifically, we set $\delta = 200$, $\pi = 5$, $\delta_{max} = 0.01$, and $trash_{max} = 10$. For both methods, we set $10 \leq K \leq 50$. We observe that our method constantly achieves 20% less distortion than NWA.

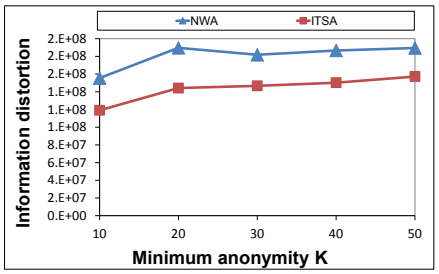


Figure 4.11: *Oldenburg*: *Information Distortion*

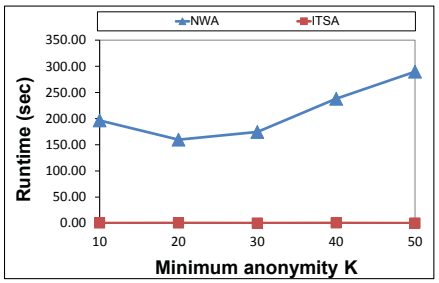


Figure 4.12: *Oldenburg*: runtime (sec)

Figure 4.12. We compare ITSA with NWA in terms of efficiency by measuring the average runtime (in seconds) of anonymizing a single window. The parameters of both methods are the same as those used in Figure 4.11, except that in this experiment we set L to be equal to the window size, $L = 5$. Increasing L requires processing longer sequences, causing our algorithm to run longer. However, Figure 4.12 shows the following two observations. First, ITSA runs significantly faster than NWA, which takes several minutes to anonymize a single window. Second, the runtime of our method is insensitive to the minimum anonymity threshold K (as also observed in Figure 4.8 (a)), steadily reporting a runtime of almost 1 second. These two observations suggest that our proposed method is suitable for anonymizing a "live" stream of trajectories.

Overall. The distortion ratio caused by ITSA is dominated by the maximum adversary's background knowledge L and the window size N. This is because both of these parameters incur larger candidate sets containing potential critical violations. This finding validates our theoretical analysis of ITSA in Chapter 4.3.4. Changing

the minimum anonymity threshold K and the inference confidence C does not cause a significant impact on distortion ratio.

4.5 Summary

Due to recent advancement in mobile technology, spatio-temporal data is being continuously generated. The data can be automatically collected by some data holder. In this chapter, we propose a novel approach for anonymizing a stream of trajectories generated by moving individuals. The anonymous trajectories are published on the fly to maintain data freshness. We illustrate and formalize two types of privacy threats. We also propose an algorithm for incrementally anonymizing a sequence of dynamically-updated sliding windows on the stream. We structure the window in a way to accommodate a massive volume of transient data. We evaluate the performance of our method on simulated and real-life datasets, and we compare with other methods. Experimental evaluation demonstrates that our method is suitable for anonymizing real-life trajectory streams, and that it outperforms existing methods.

Chapter 5

Anonymizing Static Trajectories

5.1 Introduction

Location-aware devices and systems capable of collecting user-specific data have been implemented in many smart city infrastructures over the last several years. For example, *Société de transport de Montréal* (STM), the public transport agency in Montreal, has employed a *smart card automated fare collection (SCAFC)* system to collect passenger transit data. The data collected by these systems has numerous uses for data analysis and is crucial to enactment of administrative regulations and generation of metrics to weigh business practices. However, the collected data in its raw form contains sensitive information that is specific to individuals and must be anonymized to protect their privacy before the data is shared by a certain third party for data analysis.

In this chapter, we study the problem of publishing trajectories with spatio-temporal information and propose a privacy-preserving trajectory publishing solution under the differential privacy model [34]. We assume that trajectories of moving individuals have already been collected and stored in a database system.

Let us consider the spatio-temporal transit data collected by STM. For every passenger, the collected data includes the passenger's smart card number, the

Table 5.1: Raw trajectories of 7 passengers

TID	Trajectory
tr_1	a.1 $\rightarrow$ c.2
tr_2	c.2 $\rightarrow$ b.4
tr_3	a.2 $\rightarrow$ b.3 $\rightarrow$ c.4
tr_4	c.3 $\rightarrow$ a.4
tr_5	a.1 $\rightarrow$ b.2 $\rightarrow$ c.3
tr_6	a.3 $\rightarrow$ c.4
tr_7	a.3 $\rightarrow$ b.4

visited station ID, and a timestamp. Data is collected as passengers present their unique smart cards to an automated reader for fare collection when boarding a bus or entering a metro station. Once collected, the passenger's data is stored in a central database whereby the sequence of visited stations is arranged in a timely order. Table 5.1 provides an abstract representation of the STM transit data in its raw form. Identifying pieces of information such as names and card numbers are omitted; instead, a trajectory ID has been added for ease of referencing. The sequence of $\langle location, timestamp \rangle$ pairs forms a passenger's path or *trajectory*. For example, trajectory tr_1 in Table 5.1 contains two pairs, $a.1$ and $c.2$, indicating that the passenger has visited locations a and c at timestamps 1 and 2, respectively. We note that the letters in Table 5.1 are used to represent locations, imply no order, and are not to be associated with any symbol or notation in a context other than this example or any of its continuations.

Trajectories collected by transit companies are periodically shared with internal and external organizations for various purposes, e.g., trajectory mining [116] [143] and traffic management [20] [86]. Publishing or sharing raw trajectories raises privacy concerns because the data is susceptible to attacks that rely on attacker's background knowledge about some target victims whose trajectories are included in the published data. These privacy concerns cannot be ignored because, in addition to putting passengers' privacy at risk, they can deter information sharing. We illustrate these privacy attacks in the following example.

Example 5.1.1. Suppose that Alice is a passenger whose trajectory is in Table 5.1, which is being released to the public. Possessing auxiliary knowledge about Alice, an attacker can uniquely identify Alice's full trajectory. For example, knowing that Alice was at location b at timestamp 2, an attacker can use this knowledge to associate Alice with trajectory tr_5 with 100% certainty and determine that she has traveled through location a at timestamp 1 and exited the transit system at location c at timestamp 3. Further, suppose that an attacker knows that Bob was at location b at timestamp 4. Hence, Bob's trajectory is either tr_2 or tr_7. In other words, the attacker is able to deduce that Bob was at $c.3$ with 50% confidence or at $a.3$ with 50% confidence, as well. The former attack is called *record linkage* and the latter is called *attribute linkage.* ■

Previously, publishing trajectories has been explored through syntactic and semantic privacy models. In Chapter 2.2, we discussed some syntactic privacy models, namely k-anonymity [115] and ℓ-diversity [90]. Researchers have combined these privacy models with *spatial generalization*, *space translation*, and *suppression-based* techniques to anonymize trajectories. However, while retaining data utility, these models, including our work in Chapter 4, are fundamentally prone to a number of privacy attacks [126] [48] [72]. To avoid this type vulnerability, in this chapter we propose an anonymization approach based on *differential privacy* [34].

Differential privacy is a semantic privacy model that provides provable privacy guarantees. In this chapter, we utilize differential privacy to anonymize a set of trajectories and output a sanitized version with an effective level of utility. There have been several semantic models [64] [65] [26] [61] [39] developed by researchers for achieving privacy-preserving trajectory publishing. Unlike existing approaches, however, our approach preserves the time dimension in the sanitized trajectories, allowing for more precise analysis, as opposed to preserving locations *sequentiality.*

Differential privacy works in two environments: the *interactive setting* [37] and the *non-interactive setting* [13] [130]. The former setting allows the data holder

(e.g., the transit company) to receive a limited number of requests pertaining to the underlying raw data, and sanitized answers are released. In the latter setting the data holder sanitizes then releases the entire data. In both settings, the differentially-private mechanism consumes at most ϵ budget, where ϵ is a pre-defined *privacy budget*. In this chapter, we propose a solution that works under the non-interactive setting as it gives extra flexibility in terms of analysis power, especially when there is no specific data recipient for the released data. Subsequently, the target data analysis task is unknown and can vary considerably depending on the desired analysis task. Moreover, many specific data mining techniques rely on count queries to accomplish their objective. Hence, we focus on preserving *count queries* in the sanitized data to provide various data recipients with the freedom to perform a wide variety of analysis tasks. It is worth noting that in our problem definition we assume the presence of the entire trajectory of a moving individual before sanitizing the complete set of trajectories.

Contributions. In this chapter, we investigate the problem of privacy-preserving trajectory publishing. Unlike sequential data, trajectories are spatio-temporal data that contains a time dimension. The existence of a temporal dimension renders the problem challenging because extra care is required in the sanitization process to handle the sparseness of data. We first formalize the problem of publishing trajectories under the rigorous privacy model of differential privacy in the non-interactive setting. We then propose an efficient and scalable sanitization algorithm, called *SafePath*, that models trajectories in a prefix tree structure, which significantly contributes to the ability of handling sparse and high-dimensional data [7]. The closest work to ours is by Chen et al. [26]. They proposed to sanitize *sequential data* (e.g., sequences of locations) under differential privacy and further suggested that their method could be extended to spatio-temporal data, i.e., sequences of locations *coupled* with timestamps. We implemented Chen et al.'s extension and call it *SeqPT*. Through theoretical and experimental analysis on real-life trajectory datasets, we

demonstrate that *SafePath* provides severalfold improvement over *SeqPT* with respect to efficiency in terms of runtime and scalability in terms of both number of records and dimensionality of the dataset.

5.2 Preliminaries

5.2.1 Trajectories as Prefix Tree

In this chapter, we assume that individuals are traveling from one location to another on a geographical map. The map is discretized into unique spatial areas that collectively form the location universe. A single trajectory is the trace left by a single individual, where every visited location is coupled with a timestamp. Timestamps within a single trajectory are non-decreasing and are drawn from the timestamp universe, whereas a location may appear multiple times and/or consecutively. Formally speaking,

Definition 5.1 (*Trajectory*)**.** A trajectory

$$tr = loc_1.t_1 \rightarrow loc_2.t_2 \rightarrow \dots loc_i.t_j \tag{5.1}$$

is a finite sequence of pairs consisting of a location $loc_i \in \mathcal{L}$ and a monotonically-increasing timestamp $t_j \in \mathcal{T}$, where $1 \leq i \leq |\mathcal{L}|$, $1 \leq j \leq |\mathcal{T}|$, and $t_j \leq t_{j+1}$. ■

$|tr|$ denotes the trajectory *length*, which is the number of location and timestamp pairs in tr. For example, the first trajectory tr_1 in Table 5.1 is of length $|tr_1| = 2$.

As opposed to sequential data [26], trajectory data contain a time dimension that renders the data high-dimensional and, in most cases, extremely sparse. Handling sparse data is a challenging problem because processing time is an important aspect of sanitization and should be performed in a timely manner. For

this reason, we structure a set of trajectories D as a *prefix tree*, which provides the desired compactness for achieving efficient processing. A prefix tree creates a node for every unique pair of location and timestamp, such that the node is a prefix to all its descendants. Moreover, a $root - to - leaf$ path in a prefix tree represents one unique trajectory $tr_n \in D$. Every node along a $root - to - leaf$ path contains all the trajectories in D to which the $root - to - node$ path is considered a *prefix*. Abusing notation, a trajectory $tr_m = loc_1^m.t_1^m \rightarrow loc_2^m.t_2^m \rightarrow \ldots loc_i^m.t_j^m$ is a prefix to another trajectory $tr_n = loc_1^n.t_1^n \rightarrow loc_2^n.t_2^n \rightarrow \ldots loc_i^n.t_j^n$, denoted by $tr_m \preceq tr_n$, if and only if: (1) $|tr_m| \leq |tr_n|$, and (2) $\forall loc_i^m.t_j^m \in tr_m$, $loc_i^m = loc_i^n$ and $t_j^m = t_j^n$, where $1 \leq i, j \leq |tr_m|$. For instance, let $tr_1 = loc_1.t_1 \rightarrow loc_2.t_2 \rightarrow loc_3.t_3$, $tr_2 = loc_1.t_1 \rightarrow loc_2.t_2$, $tr_3 = loc_1.t_1 \rightarrow loc_3.t_3$, and $tr_4 = loc_1.t_1 \rightarrow loc_1.t_2$. $tr_2 \preceq tr_1$, but $tr_3, tr_4 \npreceq tr_1$.

Definition 5.2 (*Prefix tree*)**.** A prefix tree $\mathbb{P} = (Nodes, Edges, Root)$ of a trajectory dataset D is a collection of $Nodes$ connected by $Edges$ and rooted at the $Root$ node. $\forall n \in Nodes$, $n(loc.t) = \langle tr(n), c(n) \rangle$, where $loc.t \in tr \in D$ is a unique pair of location and timestamp representing n, $tr(n) = \{tr \in D \mid prefix(n) \preceq tr\}$ where $prefix(n)$ is the $Root - to - n$ path, and $c(n) = |tr(n)| + \texttt{Lap}(\lambda)$ is a noisy count. The $Root$ node is represented by the pair 0.0 and $tr(Root) = D$. ■

From Definition 5.2, given a node n in a prefix tree $\mathbb{P}$, $tr(n) \subseteq D$ is the set of trajectories containing the prefix defined by the $Root - to - n$ path, symbolized as $prefix(n)$. Going from a parent node $p(loc_i.t_j)$ at level e in $\mathbb{P}$ to a child node $n(loc_{i+1}.t_{j+1})$ at level $e + 1$ implies the transition $loc_i.t_j \rightarrow loc_{i+1}.t_{j+1}$. We refer to the set of nodes at level e by $level(e)$. The $Root$ node belongs to $level(0)$, and the set of $Root$'s child nodes belongs to $level(1)$, etc.

5.2.2 Problem Statement

A trajectory dataset $D = \{tr_1, tr_2, \ldots, tr_{|D|}\}$ is a multiset of trajectories where every record in D is a trajectory that belongs to a single and unique record owner. $|D|$ denotes the size of the dataset, i.e., the number of individuals in D. A data holder has access to D and wishes to publish a differentially-private version of D, denoted by $\hat{D}$, that can be used for various data analysis tasks. In Chapter 5.3 we propose a differentially-private algorithm under the objective of maintaining *count queries* over the sanitized trajectories $\hat{D}$ with respect to the raw trajectories D.

Many data mining techniques rely on count queries to accomplish data analysis. Stemming from the goal of providing data recipients with the freedom to perform a wide variety of data analysis tasks on the sanitized trajectories, and given that the target data analysis task is unknown at the sanitization stage, we aim at sanitizing a set of raw trajectories without compromising its utility in terms of count queries.

A count query over a trajectory dataset returns the number of trajectories of which the issued query is a *subset*. Query q is a subset of trajectory tr, denoted by $q \subseteq tr$, if and only if: (1) $|q| \leq |tr|$, and (2) $\forall loc_i.t_j \in q$, $loc_i.t_j$ is also $\in tr$, where $|q|$ denotes the length of the query, i.e., the number of location and timestamp pairs in q. In contrast to a prefix, a count query ignores sequentiality of pairs. For example, suppose query $q = loc_1.t_1 \rightarrow loc_5.t_{10}$ is issued over the dataset $D = \{loc_1.t_1 \rightarrow loc_5.t_{10}, loc_1.t_1 \rightarrow loc_2.t_5 \rightarrow loc_5.t_{10}\}$. Then, the returned answer $q(D) = 2$.

Definition 5.3 (*Count query*)**.** Let q be a count query that contains a sequence of location and timestamp pairs in accordance with Definition 5.1. A count query q issued over a trajectory dataset D is defined as $q(D) = |\{tr \in D \mid q \subseteq tr\}|$. ■

The utility of a count query q over a sanitized data $\hat{D}$ is computed by its *relative error* [128] [130] [26], which measures how far the noisy answer $q(\hat{D})$ is from the true answer $q(D)$. That is, $relative\_error(q(\hat{D})) = \frac{|q(\hat{D}) - q(D)|}{q(D)}$. However, when $q(D)$ returns a very small value, the computed error becomes very large. In

order to limit the impact of extremely small count queries, it is common to use a *sanity bound* [128] [130]. This prevents the relative error from being excessively dominated by extremely small fractions of the data [56]. Therefore, the relative error is computed as follows:

$$relative\_error(q(\hat{D})) = \frac{|q(\hat{D}) - q(D)|}{max\{q(D), sanity\_bound\}}.$$

In Chapter 5.4, we choose $sanity\_bound = 0.1\%$ of the raw dataset as in [26] [130].

In this chapter, we tackle the problem of *privacy-aware trajectory publishing*. We present the following definition:

Definition 5.4 (*Privacy-aware trajectory publishing*)**.** Given a raw trajectory dataset D, a privacy budget ϵ, and a set of taxonomy trees, we wish to publicly publish $\hat{D}$, a differentially-private version of D, such that: (1) $\hat{D}$ minimizes the distortion inflicted on count queries due to sanitization, and (2) the sanitization process is efficient and scalable to handle extremely sparse trajectories. ■

5.3 Proposed Algorithm

In this section we present *SafePath*, our proposed algorithm for publishing differentially-private trajectories. Chapter 5.3.1 gives an overview of the entire algorithm. Chapters 5.3.2 and 5.3.3 describe the algorithm in detail. We end with a theoretical analysis of the proposed solution in Chapter 5.3.4.

5.3.1 Overview

Summary. We propose a sanitization algorithm primarily comprised of two phases: (1) building the noisy prefix tree, and (2) constructing the sanitized dataset. Given a raw trajectory dataset D, a privacy budget ϵ, location and timestamp taxonomy trees $\mathbb{T}_{loc}$ and $\mathbb{T}_{time}$, respectively, and the height, h, of the noisy prefix tree, our

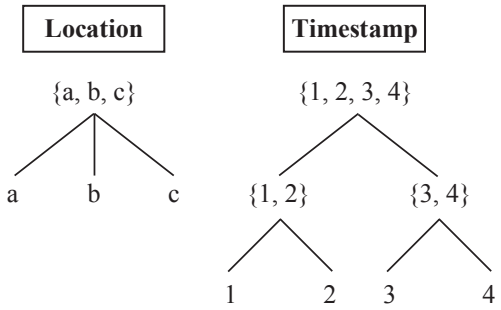


Figure 5.1: Taxonomy trees

proposed sanitization algorithm (presented in Algorithm 5.1) returns a differentially-private trajectory dataset $\hat{D}$. Algorithm 5.1 first builds the noisy prefix tree $\mathbb{P}$, and then feeds $\mathbb{P}$ to *EnhanceConsistency* that constructs $\hat{D}$ by systematically traversing $\mathbb{P}$ to ensure data consistency along trajectory paths. Lastly, Algorithm 5.1 releases the sanitized trajectory dataset $\hat{D}$.

Taxonomy Trees. A taxonomy tree consists of a generalization hierarchy in which every node contains a unique range interval that is a subset of the domain values. The range intervals of all the nodes that belong to the *same* generalization level collectively constitute the entire domain. A taxonomy tree is defined by two parameters on which we assume no restrictions: the *taxonomy tree height* and the number of *children*. The former parameter defines the height of the generalization hierarchy, and the latter parameter defines the maximum number of child nodes that belong to the same parent node in the tree. An example of user-defined taxonomy trees is given in Figure 5.1. The height of the timestamp taxonomy tree is 1 because there exists one general level, and the number of children is 2, whereas the height of the location taxonomy tree is 0 (hierarchies are ignored). In Chapter 5.4, we experimentally evaluate the impact of different taxonomy trees on the performance of our method by varying the taxonomy tree height and the number of children.

Privacy Budget Allocation. In order to satisfy ϵ-differential privacy, Algorithm 5.1 effectively distributes the input parameter ϵ among its differentially-private operations. ϵ is *uniformly* distributed among each level in the noisy prefix tree, i.e.,

every level e is allocated $\dot{\epsilon} = \frac{\epsilon}{h}$. Upon construction, a level consists of two sublevels: location and timestamp. Each sublevel is assigned an equal portion of the level's privacy budget, i.e., $\ddot{\epsilon} = \frac{\dot{\epsilon}}{2}$. Furthermore, each sublevel consists of a hierarchy that generalizes location/timestamp domain values to multiple generalization levels. Here, we employ *non-uniform* budget distribution, as follows. Abusing notation, a general node at generalization level i in taxonomy tree $\mathbb{T}$ is allocated a privacy budget $\epsilon_g = i \times \epsilon_u$, where $\epsilon_u = \frac{2\ddot{\epsilon}}{|\mathcal{U}|}$ is a *unit budget* defined as a function of the underlying universe size $|\mathcal{U}|$. A non-general node receives a privacy budget $\epsilon_{ng} = \frac{(|\mathcal{U}| - 2\sum_{x=1}^{i} x)\ddot{\epsilon}}{|\mathcal{U}|}$. We choose such an allocation scheme because less general nodes (at the lower levels of a given hierarchy) contain smaller trajectory counts, thus it is fair to increase the allocated budget portion as the nodes go deeper in the hierarchy. In Chapter 5.4, we examine the effect of ϵ on data utility.

We note that only non-general nodes will be added to the prefix tree. The sublevels in the noisy prefix tree, including general nodes, are only part of the building process.

5.3.2 Building the Noisy Prefix Tree

Our proposed algorithm builds a noisy prefix tree whereby trajectories are distributed among tree nodes based on noisy counts and prefixes. The idea is to construct tree level e by extending every node at level $e - 1$. In order to satisfy differential privacy, a differentially-private operation should not depend on the underlying dataset. Recall a node is represented by a unique pair of location and timestamp; we extend every node by considering *all* possible combinations of location and timestamp given their respective universes. In other words, any location and timestamp pair that does not appear in the raw trajectories has a *non-zero probability* of appearing in the sanitized trajectories.

Example 5.3.1. Figure 5.2 presents a running example of a noisy prefix tree $\mathbb{P}$, where the input dataset is the raw trajectories in Table 5.1. The *Root* of $\mathbb{P}$ is the

first node at the top of the tree. Any node in $\mathbb{P}$ consists of three pieces of information: location and timestamp pair(s), a portion of the dataset trajectories, and a noisy count (except the *Root*). The height of the tree is 2, and each level consists of two sublevels separated by a dashed line. ■

A node is added to the noisy prefix tree, and thus qualifies for extension, if it is considered to be *non-general* and *non-empty*. A non-general node is represented by a specific location and a specific timestamp (as opposed to a general node as described in Chapter 5.3.1). A decision whether a node is non-empty is rendered based on the node's noisy count. Recall a node n's noisy count is stored in its own $c(n)$, a non-empty node is a node with $c(n) \geq \theta$, where θ is a pre-defined threshold computed as follows. Let node $n \in level(e)$, we define the noisy count threshold θ as a function of level e's allocated budget portion, $\dot{\epsilon}$. More specifically, we define θ to be two times the standard deviation of the level's noise. Recall from Chapter 2.3.3 that the Laplace mechanism has variance $2\lambda^2$, where $\lambda = \frac{1}{\epsilon}$ for count queries as a utility function. Therefore, $\theta_{ng} = 2\frac{\sqrt{2}}{\dot{\epsilon}}$, where θ_{ng} is defined for non-general nodes. The same concept is applied to general nodes when building taxonomy trees. That is, a general node is deemed non-empty, and thus can be extended within its taxonomy tree, if its noisy count is not less than θ_g, where $\theta_g = 4\frac{\sqrt{2}}{\dot{\epsilon}}$.

Example 5.3.2. For the sake of simplicity, let $\theta_{ng} = \theta_g = 5$ for the example in Figure 5.2. Any node with noisy count < 5 (barred) is considered empty and is not going to be extended. Grey nodes indicate the nodes that are added to the noisy prefix tree. ■

The above filtering technique prunes the nodes with small true counts, albeit with reasonable impact on data utility as suggested by the experiments performed on real-life datasets in Chapter 5.4. On the other hand, nodes that contain no trajectories (i.e., $tr(n) = 0$) get filtered out in the early stages of building the noisy prefix tree. This significantly improves utility and efficiency by preventing

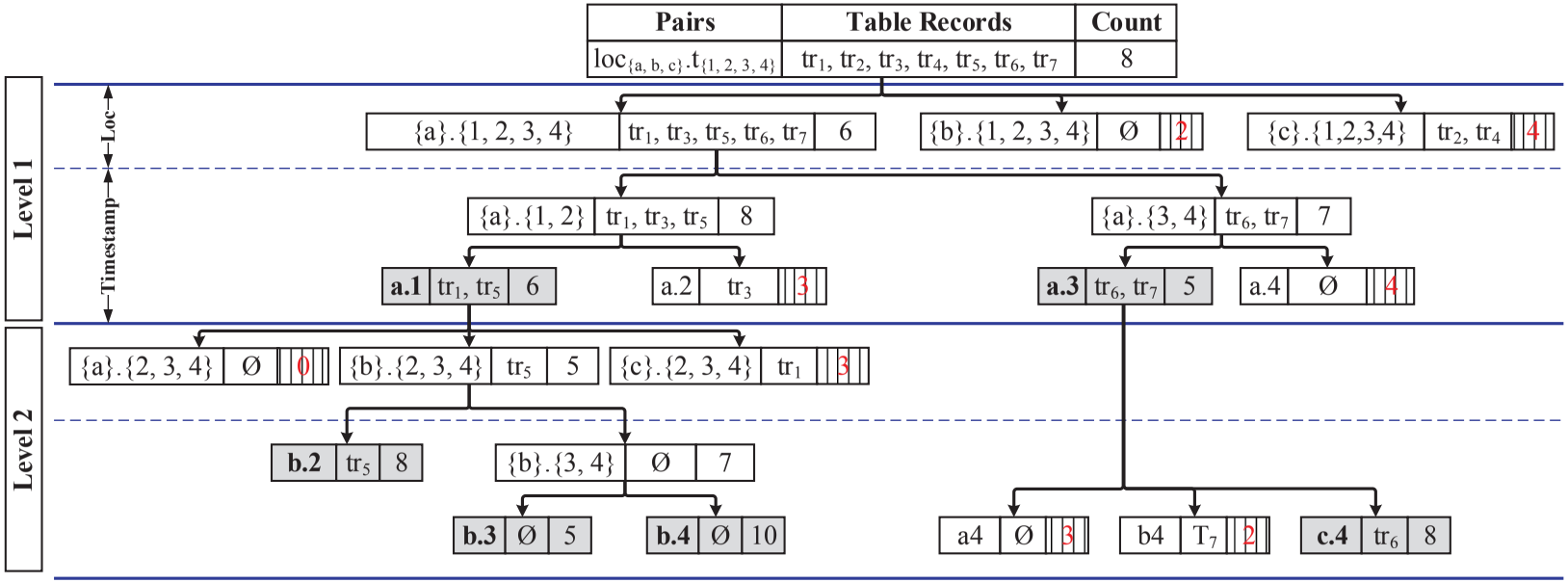


Figure 5.2: Noisy prefix tree of the trajectories in Table 5.1

Algorithm 5.1: *SafePath*

Input: Raw trajectory dataset D, privacy budget ϵ
Input: Taxonomy trees: $\mathbb{T}_{loc}$ and $\mathbb{T}_{time}$
Input: Height of the noisy prefix tree h
Output: Differentially-private trajectory dataset $\hat{D}$

1: Create a prefix tree $\mathbb{P}$ with node *Root*;
2: $tr(Root) \leftarrow$ all trajectories in D;
3: $\dot{\epsilon} = \frac{\epsilon}{h}$;
4: Compute ϵ_g and ϵ_{ng} for loc. and timestamp sublevels;
5: Compute θ_g and θ_{ng};
6: $i = 1$;
7: **while** $i \leq h$ **do**
8: **for** each non-general node in $level(e-1)$ **do**
9: $\mathcal{W} \leftarrow BuildSubLevel(\mathbb{T}_{loc}, \epsilon_g, \epsilon_{ng}, \theta_g, \theta_{ng})$;
10: **for** each non-general location node in $\mathcal{W}$ **do**
11: $level(e) \leftarrow BuildSubLevel(\mathbb{T}_{time}, \epsilon_g, \epsilon_{ng}, \theta_g, \theta_{ng})$;
12: **end for**
13: **end for**
14: $i++$;
15: **end while**
16: $\hat{D} \leftarrow EnhanceConsistency(\mathbb{P})$;
17: **return** $\hat{D}$;

building and processing false trajectories. We next describe our entire approach, as summarized in Algorithm 5.1.

Algorithm 5.1 starts at Line 1 by creating a *Root* node under which the noisy prefix tree $\mathbb{P}$ will be built. Line 2 assigns all the trajectories of the input dataset D to $tr(Root)$, and Line 3 computes the privacy budget portion dedicated for each level. Lines 4 computes the privacy budget portions ϵ_g and ϵ_{ng} for the general and non-general nodes in both the location and timestamp sublevels. We note that the location and timestamp sublevels have the same pair of budget portions ϵ_g and ϵ_{ng} if taxonomy trees $\mathbb{T}_{loc}$ and $\mathbb{T}_{loc}$ and $\mathbb{T}_{time}$ are identical (see Chapter 5.3.1). Line 5 computes the noisy count thresholds θ_g and θ_{ng}. Lines 7-12 build the noisy prefix tree by iteratively constructing every level $e \leq h$ in a breadth-first manner. Under every non-general node $n \in level(e-1)$, the algorithm attempts to extend n by

first building the location sublevel (Line 9), and then under every non-empty *non-general location node*, the algorithm builds the timestamp sublevel (Line 10). A non-general location node is a node that contains a specific location value but a general timestamp value (range interval). In Line 9, *BuildSubLevel* builds the location sublevel and returns the set of all the non-empty, non-general location nodes, *BuildSubLevel* builds the timestamp sublevel and returns the set of all the non-empty, non-general nodes. Note that only the nodes returned by *BuildSubLevel* in Line 10 are added to the noisy prefix tree $\mathbb{P}$.

Algorithm 5.2 describes *BuildSubLevel*. Line 2 calls *BuildTaxonomy* that builds a generalization hierarchy according to a user-input taxonomy tree $\mathbb{T}$, where $\mathbb{T}$ is rooted at a non-general node from the previous prefix tree level. *BuildTaxonomy* (presented in Algorithm 5.3) returns the set of *non-empty leaf general nodes*, $\mathcal{C}$. Lines 3-8 of *BuildSubLevel* iterate through all the non-general nodes in $\mathcal{C}$ to determine if they are non-empty. We note that if *BuildSubLevel* is building a *location sublevel*, then $prefix(n) \preceq tr$ in Line 5 ignores the general timestamp information in the non-general location node n because the timestamp at this point is a range interval that is by default a prefix to any timestamp value that exists within the range.

5.3.3 Constructing the Sanitized Trajectories

In Chapter 5.3.2, we detailed the steps for building a noisy prefix tree $\mathbb{P}$ that contains the differentially-private trajectories, as summarized in Lines 7-12 of Algorithm 5.1. At this point, trajectories are sanitized and ready for release. Constructing the sanitized trajectories from $\mathbb{P}$ is performed as follows. Starting from the *Root* node, we traverse $\mathbb{P}$ such that each and every node n is visited exactly once. Upon visiting node n, where $n \neq Root$, we construct $c(n)$ copies of $prefix(n)$ (see Definition 5.2) and append them to the output dataset $\hat{D}$.

Since it is likely for a set of trajectories to terminate at a parent node, a parent node's true trajectory count is never less than the sum of its children's true counts.

Algorithm 5.2: *BuildSubLevel*

Input: Taxonomy tree $\mathbb{T}$
Input: Privacy budgets ϵ_g and ϵ_{ng}
Input: Thresholds θ_g, and θ_{ng}
Output: Set of non-general location or timestamp nodes $\mathcal{W}$

1: $\mathcal{W} = \emptyset$;
2: $\mathcal{C} \leftarrow BuildTaxonomy(\mathbb{T}, \epsilon_g, \theta_g)$;
3: **for** each general node $C \in \mathcal{C}$ **do**
4: **for** each non-general node n in C's children **do**
5: $tr(n) = \{tr \in C \mid prefix(n) \preceq tr\}$;
6: $c(n) = |tr(n)| + \texttt{Lap}(1/\epsilon_{ng})$;
7: **if** $c(n) \geq \theta_{ng}$ **then**
8: $\mathcal{W} \leftarrow n$;
9: **end if**
10: **end for**
11: **end for**
12: **return** $\mathcal{W}$;

Considering that noise is added to true trajectory counts, the above rule may not hold for noisy counts. Hence, we need to make sure that $c(n)$ is no less than the sum of its children's noisy counts. Let p be a parent node in $\mathbb{P}$. We define a *consistent noisy prefix tree* as follows:

Definition 5.5 (*Consistent noisy prefix tree*)**.** Given a noisy prefix tree $\mathbb{P}$, $\forall p$ in $\mathbb{P}$, where $p \neq Root$, it holds that $c(p) \geq \sum_{n \in children(p)} c(n)$. ■

The objective of *EnhanceConsistency* in Line 13 of Algorithm 5.1 is to enforce the above condition on the noisy prefix tree $\mathbb{P}$ before releasing the sanitized trajectories. We introduce a post-processing step that modifies the noisy counts of the nodes in $\mathbb{P}$ such that Definition 5.5 is satisfied. Post-processing noisy counts has been applied in similar privacy problems whereby *consistent estimates* are computed to achieve more accurate results [60] [26]. Consequently, the objective of *EnhanceConsistency* is to find the consistent estimate for each node in the noisy prefix tree $\mathbb{P}$, except the *Root* node.

The consistent estimate, denoted by $\bar{c}(n)$, is described in terms of the noisy

Algorithm 5.3: *BuildTaxonomy*

Input: Taxonomy tree $\mathbb{T}$
Input: Privacy budget ϵ_g
Input: Threshold θ_g
Output: Set of general location or timestamp nodes $\mathcal{C}$

```
1: j = 1;
2: G = ∅;
3: while j ≤ height of T do
4:    G ← set of general nodes in T at level j;
5:    for each general node G ∈ G do
6:       tr(G) = {tr ∈ parent | prefix(G) ⪯ tr};
7:       c(G) = |tr(G)| + Lap(1/ε_g);
8:       if c(G) < θ_g then
9:          Remove G and its descendants from T;
10:      end if
11:   end for
12:   j++;
13: end while
14: C = G; // Non-empty leaf general nodes
15: return C;
```

count $c(n)$ for each node n in $\mathbb{P}$. The consistent estimate is computed by traversing $\mathbb{P}$ starting from the first level and going towards the leaves. Let p be the parent node of n and s be a sibling of n; the idea is to lower each noisy count $c(s) \in children(p)$ with respect to $\bar{c}(p)$ such that the sum of all the children's noisy counts does not exceed the parent's noisy count. The rationale behind decreasing the children's noisy counts (as opposed to increasing the parent's noisy count) stems from the fact that the sum of the children's noisy counts results in a numeric value that has $|children(p)|$ times the Laplacian noise of the parent's count. Intuitively, suppressing the extra noise helps in achieving a more accurate noisy version of the raw trajectories. The consistent estimate $\bar{c}(n)$ of node n is computed as follows:

$$\bar{c}(n) = \begin{cases} c(n), & n \in level(1) \\ c(n) + min(0, \dfrac{\bar{c}(p) - \sum_{s \in children(p)} \bar{c}(s)}{|children(p)|}), & \text{o.w.} \end{cases}$$

Lastly, the differentially-private trajectories are ready to be released. Line 14 of Algorithm 5.1 traverses the consistent noisy prefix tree in a top-down fashion. Each node n is visited exactly once, and $\bar{c}(n)$ copies of $prefix(n)$ are appended to the output dataset $\hat{D}$.

5.3.4 Theoretical Analysis

Performance Improvement. Employing a *simple* prefix tree structure does achieve a differentially-private trajectory dataset. Chen et al. [26] have suggested a similar approach towards publishing sanitized trajectories; we call it *SeqPT*. In this approach, no hierarchies are imposed; rather, each node is extended by considering every possible combination of location and timestamp. In other words, if an empty node passes threshold θ, then the subtree rooted at that node will consist solely of empty nodes. On the other hand, our proposed solution (as described in Algorithm 5.1) filters out empty nodes as early as possible, preventing false trajectories from being constructed. Subsequently, runtime is significantly reduced while data utility is greatly boosted. We present herein a formal analysis that estimates the number of empty nodes that our solution averts processing, compared with the simple prefix tree solution.

Lemma 5.1. Let $p(x) = \frac{1}{2\lambda} exp(\frac{-x}{\lambda})$ be the probability density function of the Laplace distribution. Given sensitivity $\Delta f = 1$ for a count queries-based function f, and privacy budget portion $\dot{\epsilon}$, then $\lambda = \frac{\Delta f}{\dot{\epsilon}} = \frac{1}{\dot{\epsilon}}$. Hence, $p(x) = \frac{\dot{\epsilon}}{2} exp(-x\dot{\epsilon})$. Given threshold θ, the probability of an empty node having noisy count $x \geq \theta$ is

$$Pr_\theta = Pr[x \geq \theta] = \int_\theta^\infty \frac{\dot{\epsilon}}{2} exp(-x\dot{\epsilon}) dx = \frac{1}{2} exp(-\dot{\epsilon}\theta). \blacksquare \tag{5.2}$$

Recall from Chapter 5.3.2 that $\theta_{ng} = 2\frac{\sqrt{2}}{\dot{\epsilon}}$ and $\theta_g = 4\frac{\sqrt{2}}{\dot{\epsilon}}$. From Equation 5.2, the probabilities of an empty general node and an empty non-general node passing their respective thresholds are $Pr_{\theta_g} = \frac{exp(-4\sqrt{2})}{2}$ and $Pr_{\theta_{ng}} = \frac{exp(-2\sqrt{2})}{2}$. Let empty

node n be at level e in a given prefix tree. In a simple prefix tree, the estimated number of empty nodes descending from n is $\mathbb{E}_1 = (|\mathcal{L}||\mathcal{T}|Pr_{\theta_{ng}})^{h-e}$, where $\mathcal{L}$ is the location universe, $\mathcal{T}$ is the timestamp universe, and h is the height of the prefix tree. In a noisy prefix tree, the estimated number of empty nodes descending from n, $\mathbb{E}_2$, is evaluated as follows. For simplicity, let the location and timestamp taxonomy trees be of the same height; i.e., $h_{\mathbb{T}_1} = h_{\mathbb{T}_2} = h_{\mathbb{T}}$. Further, let F denote the maximum number of general child nodes in any taxonomy tree; thus, $F^{h_{\mathbb{T}}}$ is the number of leaf general nodes. This estimates the number of empty location nodes in one location sublevel to $F^{h_{\mathbb{T}}}(Pr_{\theta_g}^{h_{\mathbb{T}}} \cdot \frac{|\mathcal{L}|}{F^{h_{\mathbb{T}}}} Pr_{\theta_{ng}}) = F^{h_{\mathbb{T}}} Pr_{\theta_g}^{h_{\mathbb{T}}} \cdot |\mathcal{L}| Pr_{\theta_{ng}}$ and the number of empty timestamp nodes in one timestamp sublevel *under one empty location node* to $F^{h_{\mathbb{T}}}(Pr_{\theta_g}^{h_{\mathbb{T}}} \cdot \frac{|\mathcal{T}|}{F^{h_{\mathbb{T}}}} Pr_{\theta_{ng}}) = F^{h_{\mathbb{T}}} Pr_{\theta_g}^{h_{\mathbb{T}}} \cdot |\mathcal{T}| Pr_{\theta_{ng}}$. Consequently, the estimated number of empty non-general nodes in one level in the noisy prefix tree is $(F^{h_{\mathbb{T}}} Pr_{\theta_g}^{h_{\mathbb{T}}} \cdot |\mathcal{L}| Pr_{\theta_{ng}}) \cdot (F^{h_{\mathbb{T}}} Pr_{\theta_g}^{h_{\mathbb{T}}} \cdot |\mathcal{T}| Pr_{\theta_{ng}}) = F^{2h_{\mathbb{T}}} |\mathcal{L}||\mathcal{T}| Pr_{\theta_g}^{2h_{\mathbb{T}}} Pr_{\theta_{ng}}^2$. Hence, $\mathbb{E}_2 = (F^{2h_{\mathbb{T}}} |\mathcal{L}||\mathcal{T}| Pr_{\theta_g}^{2h_{\mathbb{T}}} Pr_{\theta_{ng}}^2)^{h-e}$. The number of empty nodes a noisy prefix tree averts processing is estimated by the following equation:

$$
\begin{aligned}
\frac{\mathbb{E}_1}{\mathbb{E}_2} &= \frac{(|\mathcal{L}||\mathcal{T}| Pr_{\theta_{ng}})^{h-e}}{(F^{2h_{\mathbb{T}}} |\mathcal{L}||\mathcal{T}| Pr_{\theta_g}^{2h_{\mathbb{T}}} Pr_{\theta_{ng}}^2)^{h-e}} \\
&= \frac{1}{(F^{2h_{\mathbb{T}}} Pr_{\theta_g}^{2h_{\mathbb{T}}} Pr_{\theta_{ng}})^{h-e}} \\
&= \frac{1}{(F^{2h_{\mathbb{T}}} \cdot \frac{1}{2^{2h_{\mathbb{T}}} exp(-8h_{\mathbb{T}}\sqrt{2})} \cdot \frac{1}{2} exp(-2\sqrt{2}))^{h-e}} \\
&= \frac{1}{(\frac{F^{2h_{\mathbb{T}}}}{2^{2h_{\mathbb{T}}+1}})^{h-i} (exp(-2\sqrt{2}(4h_{\mathbb{T}}+1)))^{h-e}} \\
&= (\frac{2^{2h_{\mathbb{T}}+1}}{F^{2h_{\mathbb{T}}}})^{h-e} exp(2\sqrt{2}(4h_{\mathbb{T}}+1)(h-e)).
\end{aligned}
\tag{5.3}
$$

Theorem 5.1. *Given location and timestamp taxonomy trees, each having height $h_{\mathbb{T}}$ and fan-out F, a noisy prefix tree (as implemented in Algorithm 5.1) of height h reduces the number of empty nodes generated under empty node $n \in level(e)$ due to n having adequately large noisy count by $(\frac{2^{2h_{\mathbb{T}}+1}}{F^{2h_{\mathbb{T}}}})^{h-e} exp(2\sqrt{2}(4h_{\mathbb{T}}+1)(h-e))$.* ■

Privacy Analysis. Algorithm 5.1 first builds a noisy prefix tree by accessing the raw trajectories, and then invokes *EnhanceConsistency* to achieve a consistent noisy prefix tree. Building a noisy prefix tree is performed by iteratively constructing one level at a time. Each level is dedicated privacy budget portion $\dot{\epsilon} = \frac{\epsilon}{h}$, where h is the noisy prefix tree height. One prefix tree level consists of two sublevels: location and timestamp, each is dedicated $\ddot{\epsilon} = \frac{\dot{\epsilon}}{2}$. Within a sublevel, the collective budget portions $\epsilon_{ng} + \sum \epsilon_g$ dedicated for non-general and general nodes add to $\ddot{\epsilon}$; i.e., $\frac{(|\mathcal{U}| - 2\sum_{x=1}^{i} x)\ddot{\epsilon}}{|\mathcal{U}|} + \frac{2\ddot{\epsilon}\sum_{x=1}^{i} x}{|\mathcal{U}|} = \ddot{\epsilon}$, where $i \leq$ taxonomy tree height. Thus, the algorithm consumes a privacy budget that amounts to $(2 \times \ddot{\epsilon}) \times h = \epsilon$. Such a budget allocation scheme leverages sequential and parallel compositions (Lemmas 2.1 and 2.2). That is, the total budget consumed along a $Root - to - leaf$ path amounts to ϵ, whereas the total budget needed for any group of sibling nodes is equal to the same budget portion dedicated for a single node within the group.

EnhanceConsistency post-processes the differentially-private data (the noisy counts in the noisy prefix tree) without accessing the underlying raw trajectories. Consequently, *EnhanceConsistency* maintains the same differential privacy guarantees because post-processing differentially-private data has no impact on the privacy guarantees. We refer the reader to [60] for a proof.

Theorem 5.2. *Given ϵ as a user-input privacy budget, Algorithm 5.1 is ϵ-differentially private.* ■

Complexity Analysis. We analyze the runtime of Algorithm 5.1 with respect to its input parameters by first examining Lines 7-12, which build the noisy prefix tree, followed by *EnhanceConsistency*. Building the noisy prefix tree starts by building a taxonomy tree, requiring distributing trajectories from the parent node to its children. At any level in the taxonomy tree[1] there exists exactly $|D|$ trajectories in total. Therefore, a single level in the noisy prefix tree requires scanning the input

[1] Assuming the taxonomy tree is a perfect F-ary tree.

dataset $O((h_{\mathbb{T}_1} + h_{\mathbb{T}_2})|D|) = O(h_{\mathbb{T}}|D|)$ times, where $h_{\mathbb{T}_1}$ and $h_{\mathbb{T}_2}$ are the heights of the location and timestamp taxonomies, respectively, and $h_{\mathbb{T}} = max\{h_{\mathbb{T}_1}, h_{\mathbb{T}_2}\}$.

Generating nodes is a costly operation. Generating general nodes does not depend on the underlying data and is done in linear time with respect to the user-defined taxonomy. We now estimate the number of non-general nodes in the noisy prefix tree as follows. At worst, Algorithm 5.1 fails to filter out empty nodes through general nodes; i.e., all general nodes are constructed according to their taxonomies. Recall the timestamp sublevel extends the non-general location nodes in the location sublevel. Intuitively, the number of non-general nodes in the timestamp sublevel is much greater (no less at best) than that at the location sublevel. Therefore, we estimate the number of generated nodes at the timestamp sublevel to be $|D| + \nu$, where ν represents the number of empty nodes from $\mathcal{T}$ that pass the noisy count threshold θ_{ng}. ν is a random variable that follows a *binomial distribution* and can be estimated according to Theorem 5.3 [31].

Theorem 5.3. *Let n denote the total number of empty nodes, each having a success probability Pr_θ. The number of successes, ν, in a series of independent pass/fail experiments on each node follows a binomial distribution $Bin(n, Pr_\theta)$, denoted by $\nu \sim Bin(n, Pr_\theta)$.* ■

Let $\nu'_l \sim Bin(|\mathcal{L}| - |D|, Pr_\theta)$, $\nu''_l \sim Bin(|\mathcal{L}|, Pr_\theta)$, $\nu'_t \sim Bin(|\mathcal{T}| - |D|, Pr_\theta)$, and $\nu''_t \sim Bin(|\mathcal{T}|, Pr_\theta)$. The number of generated nodes at the timestamp sublevel at level $e \leq h$ is estimated to be $|D|(1 + \nu'_t + \mathbb{Z}) + (\nu'_l \nu''_t \cdot (\nu''_l \nu''_t)^{e-1})$, where

$$\mathbb{Z} = \begin{cases} 0, & e = 1 \\ \sum_{j=0}^{e-2} ((\nu''_l \nu''_t)^j [\nu'_l \nu''_t + \nu'_t \nu''_l \nu''_t]), & e > 1. \end{cases}$$

Therefore, generating non-general nodes in a prefix tree of height h runs in $O(|D|(\nu''_l \nu''_t)^h)$, where $\nu''_l \ll \mathcal{L}$ and $\nu''_t \ll \mathcal{T}$. Subsequently, building the noisy prefix tree runs in

$O(hh_{\mathbb{T}}|D| + |D|(\nu_l''\nu_t'')^h) = O(|D|(hh_{\mathbb{T}} + (\nu_l''\nu_t'')^h))$.

EnhanceConsistency does not require scanning the input dataset; rather, performs one operation, which is computing the consistent estimate for each node. This is achieved by scanning the noisy prefix tree once in a top-down fashion whereby each node is visited twice. Therefore, *EnhanceConsistency* is performed in linear time. Lastly, Algorithm 5.1 scans the consistent noisy prefix tree once starting from *Root* to release the sanitized trajectories.

In closing, Algorithm 5.1 runs in $O(|D|(hh_{\mathbb{T}} + (\nu_l''\nu_t'')^h))$ at worst. However, this is a theoretical conclusion of the worst-case scenario. In real-life situations, h is a fairly small integer and the number of generated empty nodes at level e is much smaller than $O(|D|(\nu_l''\nu_t'')^e)$ because general nodes attempt to filter out as many empty nodes as possible. In the next section, experiments on real-life datasets successfully sanitize extremely sparse trajectories within seconds.

5.4 Experimental Evaluation

In this section, we thoroughly evaluate our proposed differentially-private trajectory sanitization method, *SafePath*, using real-life trajectories. Evaluation encompasses three criteria: utility of the sanitized data, efficiency of *SafePath* in terms of runtime, and scalability for handling large datasets. Two real-life trajectory datasets are used for conducting experiments: *Bus* and *Metro*, each containing the paths of passengers traveling through the Montreal STM bus and metro transit networks, respectively. Table 5.2 contains summary statistics of each dataset, where $|\mathcal{D}|$ is the number of records in D where each record represents a trajectory of a unique passenger, $|\mathcal{L}|$ is the size of the location universe, $|\mathcal{T}|$ is the size of the time universe, $max|tr|$ is the maximum trajectory length, and $avg|tr|$ is the average trajectory length.

SafePath is implemented in C++. All the experiments herein are performed on a 64-bit Windows 7 running on an Intel Core 2 Due 2.13GHz PC with 8GB RAM.

Table 5.2: Summary statistics of the STM datasets

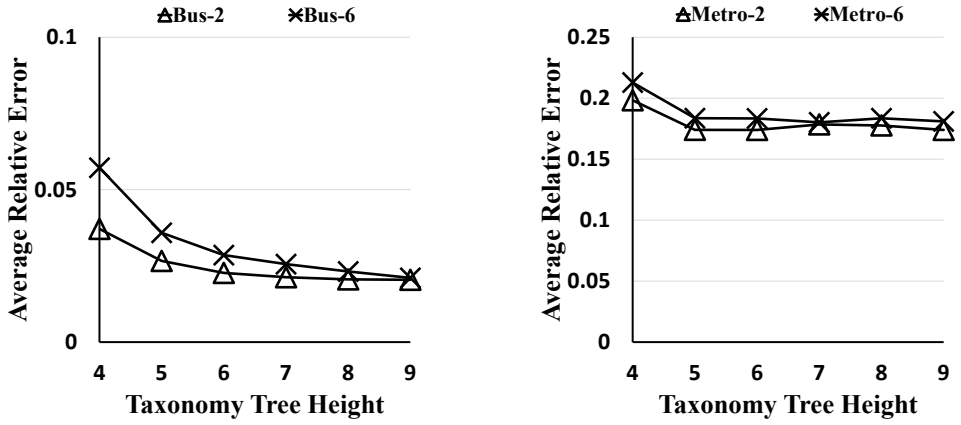
Dataset	$\|\mathcal{D}\|$	$\|\mathcal{L}\|$	$\|\mathcal{T}\|$	$max\|tr\|$	$avg\|tr\|$
Bus	773,296	893	168	121	4.69
Metro	847,668	68	168	90	3.22

5.4.1 Utility and Efficiency

We examine the impact of the different parameters of *SafePath* on utility and runtime. Similar to the evaluation methodology of [26] and [130], we measure the utility of a sanitized dataset by issuing a set of randomly generated count queries. Specifically, we issue 40,000 count queries that randomly draw values from the location and timestamp universes following an even distribution. For every query issued over the sanitized dataset, we compute its relative error, then we average all the errors and report the latter value in the graphs herein. The *length* of a count query, denoted by $|q|$, refers to the number of location and timestamp pairs in the query. For example, $q = loc_1.t_1 \rightarrow loc_2.t_2$ has length = 2. Finally, we set the sanity bound to 0.1% of the underlying dataset.

Taxonomy Trees. We examine the impact of location and timestamp taxonomy trees on utility and runtime. A taxonomy tree is defined by two parameters: *taxonomy tree height*, and number of *child nodes* (or simply *children*). The former parameter defines the number of levels in the tree, whereas the latter parameter refers to the maximum number of child nodes that belong to the same parent node. Both parameters can either be public knowledge (e.g., a metro map) or user-defined. Without loss of generality, one taxonomy tree setting will be applied to both location and timestamp hierarchies.

To demonstrate the impact of taxonomy trees on relative error, Figure 5.3 reports the average relative errors on *Bus* and *Metro*. We set the query length $|q| = h = 2$ and choose the taxonomy trees' number of child nodes to be 2 and 6, denoted in the graphs by *Bus-2*/*Metro-2* and *Bus-6*/*Metro-6*, respectively. Figure 5.3 suggests that the relative error decreases as taxonomy tree height increases and the number of

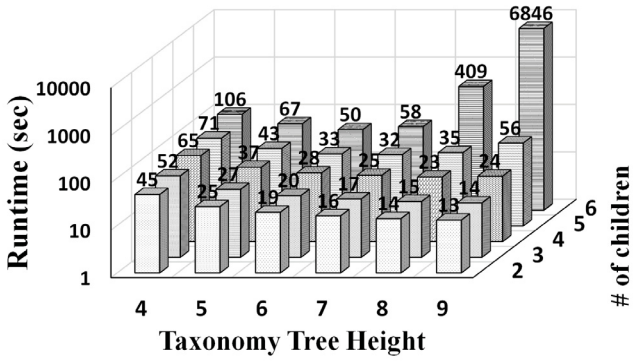


(a) *Bus* (b) *Metro*

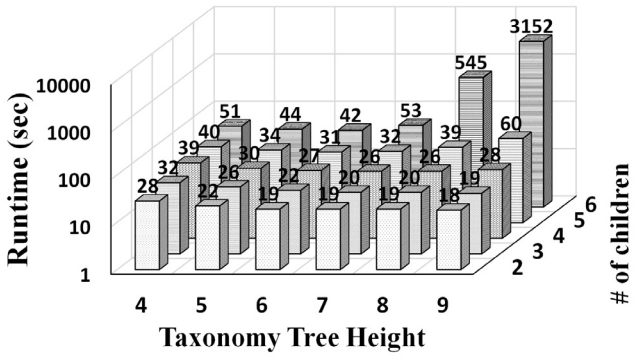
Figure 5.3: Average relative error vs. taxonomy tree height

child nodes decreases. This is because fewer empty non-general nodes are generated throughout the process of building the prefix tree, where empty nodes get filtered out by the taxonomy trees. For taxonomy tree height ≥ 6 the relative error does not exhibit significant change. Moreover, we observe that on both datasets no significant drop in error results from decreasing the number of child nodes.

Figure 5.4 shows how runtime varies under different values of taxonomy tree height and the number of child nodes. Figure 5.4(a) and Figure 5.4(b) study *Bus* and *Metro*, respectively, where the noisy prefix tree height $h = 2$, the privacy budget $\epsilon = 1$, and the runtime axis is set on a logarithmic scale. Figure 5.4 suggests that runtime increases as the number of children increases for all values of taxonomy tree height. This is because larger values of children allow for more empty general nodes to be mistakenly considered non-empty due to the added noise, resulting in a much larger number of empty non-general nodes that require extra processing. Moreover, we observe that runtime decreases non-monotonically as the taxonomy tree height increases. More specifically, runtime decreases for a smaller number of children, less than 4, after which it starts increasing up to a significant value at taxonomy tree height = 9 and number of children = 6. The increase in runtime for a larger number



(a) Bus



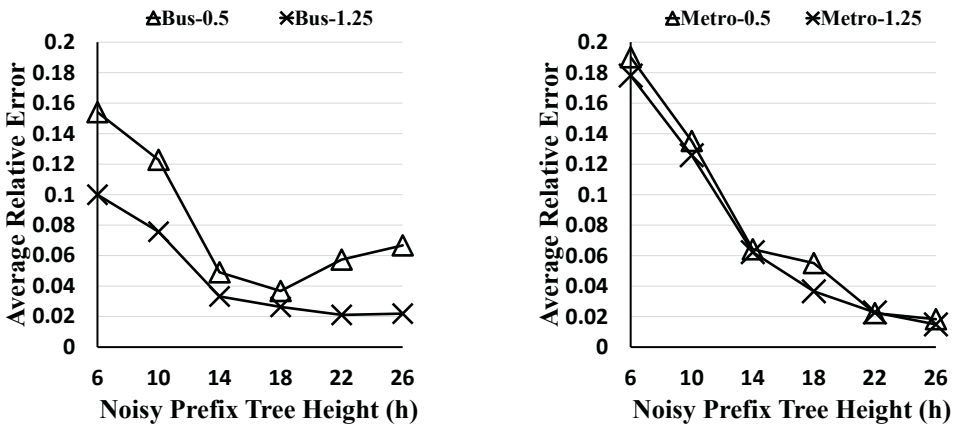
(b) Metro

Figure 5.4: Runtime vs. taxonomy tree

of children can be noticed at taxonomy tree height ≥ 6. These observations are shared by both *Bus* and *Metro*.

Figures 5.3 and 5.4 suggest that a certain range of taxonomy trees allows for a major improvement in terms of runtime without compromising utility. Unless otherwise mentioned, all our subsequent experiments will have taxonomy tree height $= 6$ and children $= 2$.

Noisy Prefix Tree Height and Privacy Budget. Figure 5.5 examines the effect of varying the noisy prefix tree height, h, and the privacy budget, ϵ, on the relative error on both *Bus* and *Metro*. The parameters are set as follows: $6 \leq h \leq 26$, $|q| = h/3$, and $\epsilon = 0.5$ and 1.25 denoted in the graph by *Bus-0.5*/*Metro-0.5* and



(a) *Bus* (b) *Metro*

Figure 5.5: Average relative error vs. noisy prefix tree height

Bus-1.25/*Metro-1.25*, respectively. Figure 5.5 reports a general decrease in the relative error when h increases. Increasing h allows us to retrieve and sanitize more data from the raw trajectories, however, up until a certain threshold, after which the added noise at each level of the noisy prefix tree becomes more dominant, causing the relative error to increase. This is more evident in sparse datasets with low privacy budget (*Bus-0.5* in Figure 5.5 (a)) because more empty nodes are generated upon building the noisy prefix tree to satisfy differential privacy. Lastly, increasing ϵ successfully manages to reduce the relative error for all values of h. It is interesting to see that our proposed method is capable of reducing the relative error below 1% on both datasets, suggesting that *SafePath* maintains high utility even for extremely sparse data.

5.4.2 Comparisons

In [26] Chen et al. proposed a prefix tree-based method for sanitizing sequential locations under differential privacy. Further, they suggested that their method can be extended to trajectories - series of locations *coupled with* timestamps. To do so, we implemented their suggested extension that exhaustively considers every possible

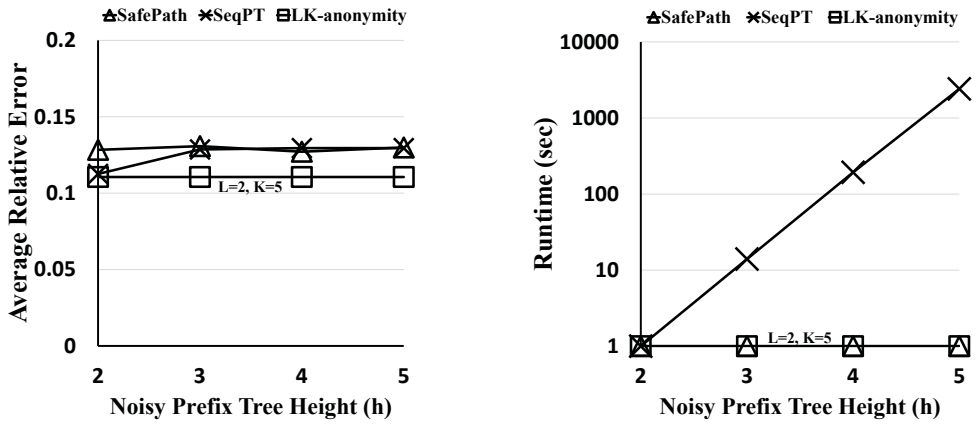
combination of location and timestamp under every node throughout the process of building the prefix tree. We call their extension *SeqPT*. Moreover, we implemented LK-anonymity [49], a syntactic-based privacy approach that makes sure that every L-sized sequence of location and timestamp pairs from the input dataset is shared by at least K trajectories via means of suppression [117] [30].

We carry out performance evaluation with respect to average relative error and runtime. *SeqPT* takes nearly 11 minutes to sanitize *Metro* when setting the prefix tree height $h = 2$, and fails to complete a single run on *Metro* when $h \geq 3$ and on *Bus* when $h \geq 2$. This is because the number of generated nodes increases exponentially as h increases. For this reason, we reduce the dimensionality of *Bus* to a maximum of 20 locations and 24 timestamps. A pair that does not fall within the shrunken universe will be excluded from the dataset. This results in a dataset with 200,000 records and a maximum trajectory length of 8. For this set of experiments we generate 10,000 non-empty random queries of size 2, set the privacy budget $\epsilon = 1$, and vary the prefix tree height $2 \leq h \leq 5$. For LK-anonymity, we choose $L = 2$ (because the average trajectory length is 1.5) and $K = 5$.

Figure 5.6 studies the performance of *SafePath*, *SeqPT*, and *LK-anonymity* in terms of utility in Figure 5.6 (a) and runtime in Figure 5.6 (b), where the runtime axis is set on a logarithmic scale. We notice that the utility achieved by *SafePath* is comparable to that achieved by *SeqPT* and *LK-anonymity* for all values of h. On the other hand, *SafePath* substantially lowers runtime from 2,400 seconds to 1 second at $h = 5$.

5.4.3 Scalability

We study the scalability of our approach by varying the *size* of the input dataset. In this set of experiments, a dataset size is determined by *both* the universe size and the number of records. We limit the timestamp universe $\mathcal{T}$ of *Bus* and *Metro* to a threshold $\leq |\mathcal{T}|$, and only consider the portion of the trajectories that satisfy



(a) Average relative error vs. h (b) Runtime vs. h

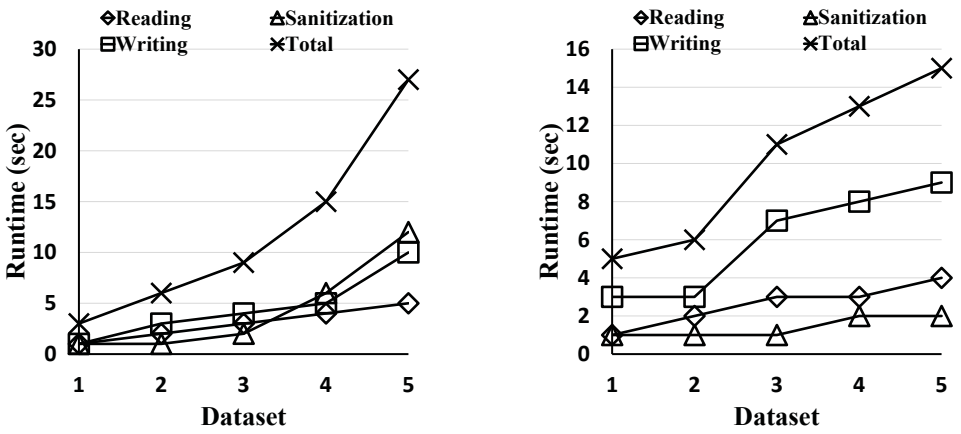
Figure 5.6: *SafePath* vs. *SeqPT* vs. *LK-anonymity*

Table 5.3: Summary statistics of sub-datasets

	$\|\mathcal{T}\|$		$\|\mathcal{D}\|$		$max\|tr\|$		$avg\|tr\|$	
Dataset	*Bus*	*Metro*	*Bus*	*Metro*	*Bus*	*Metro*	*Bus*	*Metro*
1	35	35	304,238	263,303	21	16	1.96	1.58
2	70	70	508,283	485,049	44	35	3.12	2.28
3	105	105	618,238	606,917	70	51	3.98	2.81
4	140	140	722,786	766,419	106	73	4.46	3.11
5	168	168	773,296	847,668	121	90	4.69	3.22

the timestamp limit. By varying the threshold, we achieve different datasets with smaller timestamp universes and number of records than D. We consider limiting only the timestamp universe because, unlike the location universe, the former defines the length of trajectories in the dataset. Table 5.3 contains summary statistics of 5 datasets generated from *Bus* and 5 datasets generated from *Metro*. All the scalability experiments are reported in Figure 5.7, where $h = 12$ and $\epsilon = 1$.

Figure 5.7 measures how runtime varies as we change the timestamp universe size $|\mathcal{T}|$ and number of records $|D|$ of the input raw dataset. Each value on the x-axis in Figures 5.7 (a) and (b) represents a sub-dataset extracted from *Bus* and *Metro*, respectively, as illustrated in Table 5.3. Runtime increases reasonably with the increase of the input dataset size. We observe that sanitization time increases faster



(a) *Bus* (b) *Metro*

Figure 5.7: Runtime vs. $|\mathcal{T}|$ and $|D|$

on the *Bus* sub-datasets because *Bus* has a larger location universe size compared with *Metro*; hence, more processing of empty nodes is required. In total, our method takes 27 seconds to process the full *Bus* dataset and 15 seconds to process the full *Metro* dataset.

Overall. Our proposed method achieves high data utility for count queries and is robust to handle large-scale and extremely sparse trajectory data without compromising runtime. Such robustness stems from the ability to fine-tune taxonomy trees according to the desired utility. In order to achieve a reasonable balance between utility and runtime, experiments on taxonomy trees suggest minimizing the number of children and choosing a moderate taxonomy tree height.

5.5 Summary

We study the problem of publishing sanitized trajectories under the rigorous model of differential privacy. Trajectories are spatio-temporal data characterized by being high-dimensional and sparse due to the existence of a time dimension; hence, extra care is required in the sanitization process in order to handle such data efficiently.

We propose *SafePath*, an efficient and scalable sanitization method for publishing differentially-private trajectories. *SafePath* structures trajectories as a noisy prefix tree, then performs a post-processing step that enhances the utility of the sanitized data. The authors of [26] proposed sanitizing sequential data and suggested that their method can be extended to trajectories. We implement their method and provide theoretical and experimental analysis on real-life trajectory datasets. Evaluation suggests that applying the above extension to trajectories fails to provide a scalable solution, whereas our proposed method demonstrates significant improvement in terms of efficiency and scalability with comparable data utility.

Chapter 6

Anonymizing Relational Data

6.1 Introduction

Data gathering has been witnessing an exponential growth thanks to modern advancement in information technology. The possession of collected data gives power to the data holder by enhancing data analysis and aiding in decision making. Examples include government agencies collecting census data for demographic analysis to provide better social services; transportation authorities collecting trajectories for traffic analysis to enhance the city's transportation network; hospitals collecting patients' symptoms for better future diagnosis; and online service providers collecting online surfing habits for building and enhancing recommendation systems.

As demonstrated in Chapter 1, there are cases where the data holder may not always have the expertise to perform the required data analysis [57], or the collected data must be published as mandated by the law [23]. Consequently, data publishing has become a common practice for the mutual benefit of the data holder and the data recipient. It is of no less importance that the privacy of individuals whose data is being published should be safe-guarded. To bridge the gap between these two seemingly conflicting requirements, several privacy models have been proposed in the literature.

In Chapter 2, we discussed some widely-used syntactic privacy models, whereby the output dataset has to comply with a syntactic privacy requirement. A prime example is k-anonymity and its various extensions [115] [110] [84] [89] by which every record in the output dataset has to be hidden within a group of k records. Syntactic privacy, though effective, only reduces the possibility of privacy attacks by making certain assumptions about an attacker's background knowledge about the individuals in the published data. Moreover, syntactic privacy is inherently prone to certain attacks, such as *minimality attack* [126], *composition attack* [48], and *deFinetti attack* [72].

Differential privacy [35] [36], presented in Chapter 2.2.5, is a probabilistic privacy model that provides provable privacy guarantees and works independently of any attacker's background knowledge. Intuitively, differential privacy ensures that the output of any analysis performed on the published data does not overly depend on any single participant. Individuals' privacy is protected in the sense that an individual's participation (or withdrawal) from the collected data would not significantly change the outcome of the analysis. This indirectly removes the privacy concerns of both the participants and the data holders. Therefore, in this chapter we adopt differential privacy in the non-interactive setting as our privacy model.

Figure 6.1 illustrates an overview of a privacy-preserving data publishing scenario under the non-interactive setting. In general, the data holder collects data structured in a *relational form* from a group of individual and wants to release the collected data to a third-party for data analysis without compromising the individuals' privacy. The research problem studied in this chapter is summarized by this question: how to convert the raw data to a differentially-private version via *multidimensional generalization* while maintaining utility in the published data?

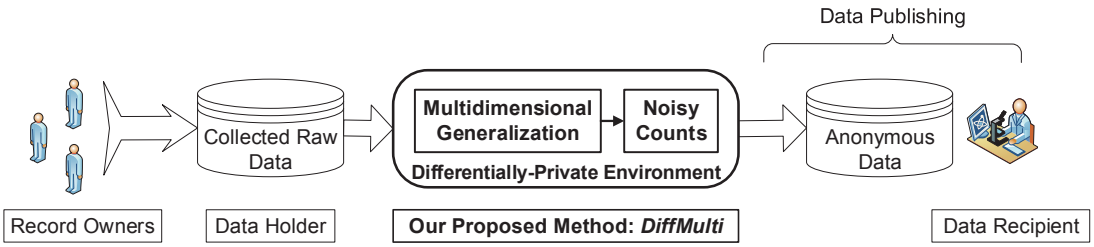


Figure 6.1: Data publishing under the non-interactive setting

6.1.1 Motivation

Methods based on the non-interactive setting mainly rely on publishing a noisy version of the contingency table of the raw dataset [13] [33] [107] [130] [31]. In other words, the true count of every possible combination of the domain values on every attribute is first generated, and then a noise is added to the count to satisfy differential privacy. Finally, all noisy counts are published. Although the published data is differentially-private, we reason that this approach yields extremely distorted data. If the collected data is high-dimensional, then the true counts diminish to very small numbers, making the added noise very large in comparison. In this Chapter, we do not adopt this approach as it yields highly distorted data that is far from being useful for any further analysis. We further validate this observation when we experimentally evaluate our method in Chapter 6.4.

In Chapter 2.3.2, we introduced *generalization* as a commonly-used anonymization technique in the literature. We transform an input dataset by generalizing raw values to less specific yet semantically consistent values, as dictated by a pre-defined generalization hierarchy. The more the raw data is generalized, the higher distortion the original data has experienced. The literature has defined two types of generalization approaches: *single-dimensional* and *multidimensional*. In this Chapter, we enforce multidimensional generalization, backed up by the study in [76] that suggests multidimensional generalization significantly improves the utility of the anonymous data over the single-dimensional approach. We note that the previous

Table 6.1: Raw data table

Continent	**YoB**	**Class**
North_America	1947	Y
Australia	1953	Y
North_America	1955	N
Europe	1957	N
Asia	1959	N
North_America	1968	Y

work in [76] achieves multidimensional generalization by applying the k-anonymity privacy model, whereas we adopt differential privacy as our privacy model. We reason that it is possible to achieve high-quality generalized data using differential privacy. The following is an illustrating example.

Example 6.1.1. The *Population Data BC* (*PopData*)[1] is a not-for-profit organization that collects health-related data from a variety of sources with the hope of harnessing such data for the advancement of human well-being. *PopData* does not possess the means to conduct research on the collected data; therefore, it offers data sharing among researchers *upon request*. Even though patient-specific identifiers, such as Name and Address, are removed from the published data, data sharing is a lengthy process heavily based on trusting the requester (i.e., data recipient). In an attempt to increase the benefits of the collected data, we propose an alternative data-sharing solution that is not based on trust, thus allowing for a wider range of data recipients. Our proposed solution produces high-quality anonymous data and guarantees patients' privacy according to the differential privacy model.

Let Table 6.1 represent a group of patients' raw data. The table has one categorical attribute, *Continent*, and one numerical attribute, YoB (Year of Birth). Moreover, let the last attribute be a *Class* attribute with two domain values Y and N that indicate whether a patient has a chronic disease. Figure 6.2 shows two taxonomy trees that represent the domain hierarchy of attribute *Continent* and

[1]https://www.popdata.bc.ca/

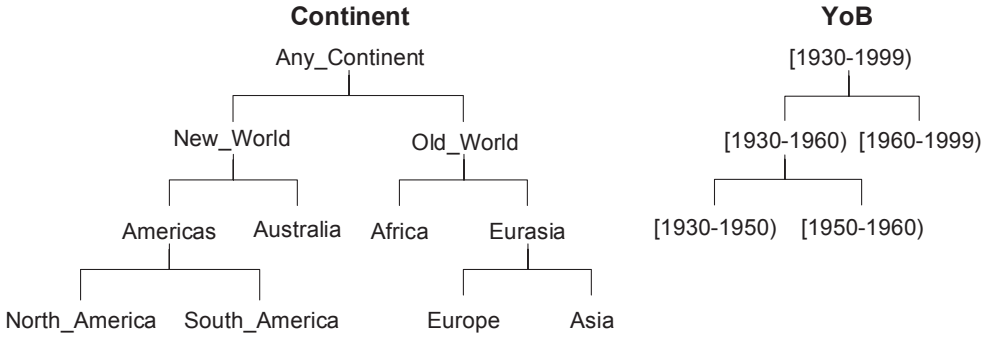
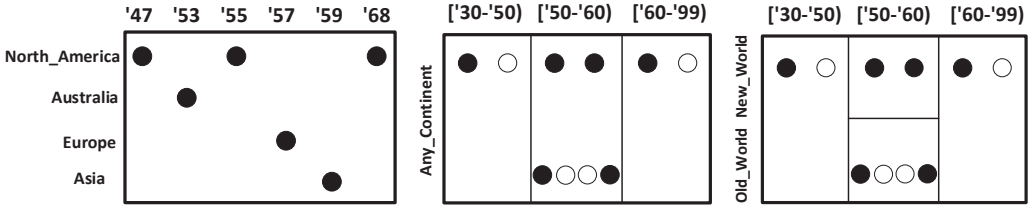


Figure 6.2: Taxonomy trees



(a) Raw data (b) Single-dimensional (c) Multidimensional

Figure 6.3: Spatial representations of Table 6.1 and its diff-priv generalizations

YoB, respectively. Figures 6.3 (a)-(c) depict a spatial representation of Table 6.1 and its two generalized versions, respectively. Figure 6.3 (c) is the result of applying our anonymization method.

Every rectangle in Figures 6.3 (b) and (c) represents a generalization region in the domain space, and every solid circle represents a record from Table 6.1. Without loss of generality, let the added noise to every region due to differential privacy be equal to 1 or 0. The added noise represents a synthetic record, and is indicated by an empty circle. Under single-dimensional generalization, *all* the raw values on the *Continent* attribute have been generalized to the topmost general value *Any_Continent*. On the other hand, the multidimensional approach generalizes *some* of the raw values to *Any_Continent* while other raw values have been replaced with more specific hierarchical values, namely, New_World and Old_World. Thus, the multidimensional approach inflicts less data distortion by producing less general data than the single-dimensional approach. ■

6.1.2 Contributions

In this Chapter, we investigate a differentially-private multidimensional solution for relational data release. To our knowledge, this is the first work to propose a concrete differentially-private algorithm that employs multidimensional generalization for relational data release in the non-interactive setting [8]. Previous related endeavors have been proposed for publishing high-quality anonymous data [96] [76] [106] [141]. Single-dimensional generalization has been used in [96] to publish differentially-private data. However, we reason that a multidimensional approach greatly increases data utility, as suggested by [76]. Even though the proposed work in [76] is a multidimensional partitioning approach, the guiding privacy model is k-anonymity, which is susceptible to syntactic-based privacy attacks [126] [48] [72]. We argue that stronger privacy guarantees, i.e., through differential privacy, can be achieved without compromising data utility. The work in [106] proposed a partition-based general framework capable of publishing differentially-private data; however, it does not scale for datasets with a large number of dimensions [108]. Other techniques [141] utilize Bayesian networks to release differentially-private data that approximate the distribution of the high-dimensional input data. In this Chapter, we evaluate the performance of our proposed algorithm in terms of data utility, efficiency, and scalability. We experimentally compare between our proposed method and the aforementioned techniques; results range from our method producing comparable results at worst to performing by an order of magnitude better at best. We summarize our contributions as follows:

1. We utilize a top-down specialization approach that provides efficient multidimensional partitioning of regions in the domain space. Given a region to be further partitioned, this approach provides direct access to the records in that region without having to scan the entire data. This property adds a scalable aspect to our method when anonymizing large datasets.

2. We argue that data records are unlikely to be evenly distributed across the domain space. Hence, our proposed method gives more attention to dense regions in order to yield less abstract data for more accurate analysis of the output differentially-private data.

3. Our proposed method performs multidimensional specialization in a differentially-private way on *both* categorical and numerical attributes. Depending on the target workload, a carefully-selected categorical value is split in accordance with a pre-defined taxonomy tree, whereas a proper numerical split point is dynamically determined for every multidimensional region throughout runtime. This releases the data holder from the burden of performing extensive preprocessing.

4. We carry out extensive experiments and compare with closely-related methods from the literature. Results suggest that our proposed multidimensional algorithm is capable of significantly improving data utility without compromising the rigorous requirement of differential privacy.

6.2 Problem Definition

6.2.1 Generalization

Generalization is the act of replacing a data value with a less semantically specific version according to a pre-defined generalization hierarchy. Let D be an input dataset defined over a set of attributes $\mathcal{A} = \{A_1, \ldots, A_d\}$, where d is the number of attributes. We denote the domain of attribute A_i by $\Omega(A_i)$. *Single-dimensional generalization* is defined by a function $\phi_i : a \rightarrow g$ for *each* attribute $A_i \in \mathcal{A}$, where $a \in \Omega(A_i)$ and g is a value in the pre-defined generalization hierarchy. As a result, if a value a is chosen for generalization, all instances of a in the dataset D will be generalized as well.

To preserve more information, we employ *multidimensional generalization*. In this setting, a *vector of values* is considered for generalization instead of considering a single value at a time. The idea is to divide the d-dimensional domain space of D into non-overlapping generalization regions. Every region R_i contains a set of generalized records from D, where every raw record is uniquely mapped to its corresponding region. By that intuition, a region can be considered as an equivalence group (or a QID group as introduced in Chapter 2) because every region contains a disjoint subset of generalized records from D.

Definition 6.1 (*Multidimensional generalization*)**.** Given a raw dataset D defined over a set of attributes $\mathcal{A} = \{A_1, \ldots, A_d\}$, and a set of taxonomy trees $\{\mathbb{T}_1, \ldots, \mathbb{T}_d\}$, multidimensional generalization is defined by a *single* global function $\phi : \Omega(A_1) \times \ldots \times \Omega(A_d) \rightarrow \{R_1, R_2, \ldots, R_m\}$ that maps an entire record in D to its corresponding generalization region R_i, where the d-dimensional domain space of D is divided into disjoint generalization regions $R_1, R_2, \ldots, R_m$. ■

According to the above definition, generalizing a raw value a in a given record entails considering the entire vector of raw values in that record. In other words, given a group of records that share a raw value a on attribute A_i, the combination of raw values on every other attribute A_j, where $i \neq j$, in a given record will determine the value g to which a will be generalized.

Example 6.2.1. Consider Table 6.1 and its anonymous version under multidimensional generalization, as spatially represented in Figure 6.3 (c). Under multidimensional generalization, $North\_America$ in records $\{North\_America, 1947\}$ and $\{North\_America, 1968\}$ has been generalized to $Any\_Continent$, whereas $North\_America$ in record $\{North\_America, 1955\}$ has been generalized to a less general value, i.e., New_World. ■

Given any region R_i, each dimension represents a generalized value g of the underlying subdomain. We propose an algorithm capable of handling both categorical

and numerical attributes. For categorical attributes, g is drawn from the generalization hierarchy as specified by the taxonomy trees. For numerical attributes, on the other hand, the algorithm will adaptively select a binary split point by which a division on the continuous domain results in two disjoint intervals. The process of generating generalization regions is called *partitioning*. A good partitioning strategy is essential for improving the utility of the anonymous data. More details are discussed in Chapter 6.3.

We mentioned in Chapter 2.2.5 that all operations under ϵ-differential privacy must be insensitive to the underlying data. In other words, given two neighboring datasets D and D', where $|D \triangle D'| \leq 1$, a differentially-private algorithm must bound the probability of obtaining the same output dataset in accordance with Definition 2.1. At the same time, the algorithm must find a proper generalization function $\phi : \Omega(A_1) \times \ldots \times \Omega(A_d) \rightarrow D^d$ depending on the expected workload of the published data. Using a unified ϕ for all data values still produces ϵ-differentially-privacy outcome, however, the output data will not be tailored to the expected analysis[2]. As a result, the data utility will drop significantly, rendering any analysis on the output data useless. In Chapter 6.3, we present our algorithm for performing effective partitioning to maximize data utility without violating the rigorous requirement of ϵ-differential privacy.

6.2.2 Problem Statement

We informally describe our problem as follows: a data holder is in possession of a dataset D that contains a multiset of records, where each record belongs to a unique individual. All identifiers, such as Name and SSN, have been removed. D is defined over a set of d attributes $\mathcal{A} = \{A_1, \ldots, A_d\}$ that can also be found, partially or

[2]Unless performed randomly, having a fixed generalization function ϕ is a non-trivial task. The domain space of ϕ is as large as the cardinality of the input dataset. Moreover, the codomain of ϕ is a set of d-dimensional regions, each bounded by either an interval or a value from the generalization hierarchy. Our proposed algorithm effectively partitions the regions to maintain data utility.

entirely, in a non-sanitized, publicly available dataset containing a group of records that belong to the same individuals whose records are in D. In addition, D contains a *Class* attribute A^{cls} that is used for classification analysis. We assume that $A_i \in \mathcal{A}$ is either categorical or numerical and A^{cls} is categorical. Finally, we assume that for each categorical attribute A_i, a taxonomy tree $\mathbb{T}_i$ is provided that defines the hierarchy of values in $\Omega(A_i)$. We do not require a taxonomy tree for numerical attributes as it requires an extensive preprocessing of D to determine appropriate splitting values. Rather, our proposed algorithm performs this task adaptively upon runtime.

A data holder wishes to publish $\hat{D}$, an ϵ-differentially-private version of D, for either general data analysis or classification analysis. We present the following definition for the problem studied in this chapter.

Definition 6.2 (ϵ-*Differentially-private multidimensional generalization*)**.** Given a raw dataset D, a set of taxonomy trees, and a privacy budget ϵ, we wish to produce an ϵ-differentially-private version of D, $\hat{D}$, by means of multidimensional generalization in order to improve the utility of $\hat{D}$. ■

We argue that finding an optimal solution is not possible mainly for the following reason. The added Laplacian noise, which draws from the set of real numbers, results in an infinite number of potential solutions, making the resulting output unverifiable [100]. Thus, in the next section we propose a heuristic based on a greedy recursive splitting of domain space.

6.3 Anonymization Algorithm

We present *DiffMulti*, a *diff*erentially-private algorithm that employs *multi*dimensional generalization for relational data release. Our proposed algorithm is comprised of two phases, as illustrated in Figure 6.1. We first provide an overview, then proceed to discuss the major operations in each phase. Particularly, Chapters 6.3.2 and

6.3.3 detail Phase 1, and Chapter 6.3.4 details Phase 2. Our algorithm is outlined in Chapter 6.3.5, followed by a discussion.

6.3.1 Overview

We propose an anonymization algorithm that consists of two phases. In *Phase 1*, all the records of the input dataset D are generalized over the set of attributes $\mathcal{A} = \{A_1, \ldots, A_d\}$ in a differentially-private manner. When a group of records is generalized to the same set of values, they form a unique *equivalence group*. Then, Phase 1 recursively specializes every equivalence group. In *Phase 2*, our algorithm publishes noisy record counts pertaining to the final equivalence groups, where records are least generalized. A straightforward way to implement this method is by scanning the entire raw records to determine a split attribute, then scanning the entire records again to specialize and split them into equivalence groups. We note that in order to specialize the records of an equivalence group, it is sufficient to scan those records apart from the rest of the dataset records. Performing specialization by applying the straightforward solution gives rise to the scalability issue, wherein scanning the entire data records becomes exhaustive for very large datasets. We utilize an efficient tree data structure that provides data access on the granular level, as illustrated in Figure 6.4. A node in this tree is referred to as a *partition*. For every value in the *Class* attribute A^{cls}, a partition P_i maintains a generalized record and a pertinent *Count* that refers to the number of raw records generalized to the same attribute values and that share the same *Class* value. Record scanning is, thus, confined to the *size* of a single partition. The size of partition P_i is determined by the number of records in the partition, i.e., $|P_i| = \sum_{x=1}^{|\Omega(A^{cls})|} Count_x$.

DiffMulti, detailed in Algorithm 6.1, performs a sequence of *specializations* on the data records, as follows. The algorithm starts by creating a root partition, P_i, that contains all raw records generalized to the topmost values in the domain hierarchy on every attribute in $\mathcal{A}$. At this point, publishing a Laplacian-noisy version of

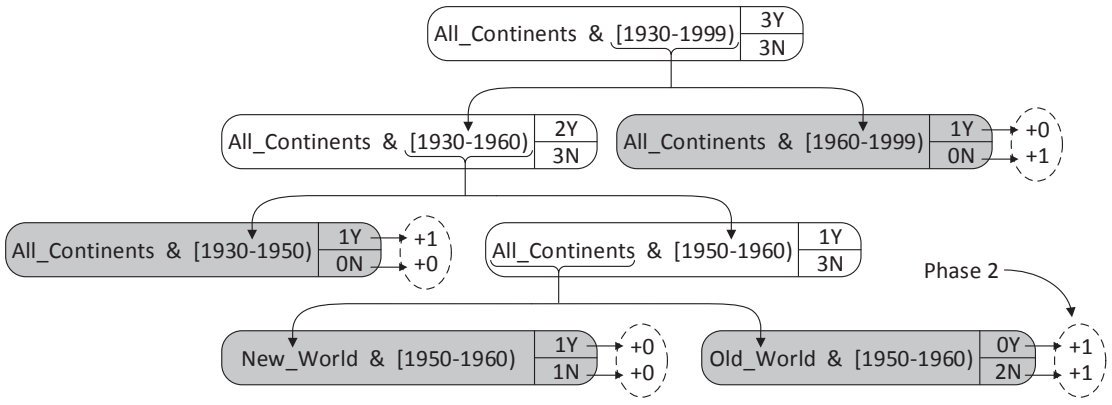


Figure 6.4: Raw data records structured as tree of partitions

$|P_i|$ guarantees ϵ-differential privacy; however, the data records are at a high level of abstraction, and analysis on such general data is far from accurate. We reduce data generality by *specializing* P_i. The maximum number of specializations is a user-input parameter to our algorithm, h. When a specialization takes place, denoted by $g \rightarrow child(g)$, a general value g gets replaced by its less general child values $child(g)$ from the domain hierarchy. In our tree data structure, every specialization creates new disjoint child partitions, each holding a set of records that generalize to the same values. A child partition $P_{c_j} \in \{P_{c_1}, \ldots, P_{c_v}\}$ represents a distinct region R in the domain space of D. Unlike single-dimensional partitioning, which performs specializations globally among all data records, our method employs multidimensional partitioning by which only a certain set of records are affected by $g \rightarrow child(g)$. In other words, a single-dimensional partitioning strategy splits across all regions in the domain space of D, while our method performs a split $g \rightarrow child(g)$ only within one region. This step is advantageous for data utility because at the end of Phase 1 less generalization is imposed on the raw records.

Example 6.3.1. We continue from Example 6.1.1. The root partition in Figure 6.4 contains all the data records in Table 6.1 generalized to the topmost values, i.e., *Any_Continent* and $[1930 - 1999)$, in addition to the true counts of records having the same value on the *Class* attribute. For simplicity, let us assume that $[1930-1999)$

is chosen for specialization. Consequently, the root partition is specialized on the YoB attribute, i.e., $[1930-1999) \rightarrow \{[1930-1960), [1960-1999)\}$. A child partition is created for every child value of $[1930-1999)$, and *DiffMulti* iteratively specializes every new partition in the tree until a leaf partition is reached (Definition 6.3). Leaf partitions are colored in grey for ease of presentation. ■

6.3.2 Choosing a Candidate

As mentioned in Chapter 6.3.1, a specialization $g \rightarrow child(g)$ replaces g with its child values from the domain hierarchy associated with an attribute A. Our algorithm greedily chooses a value g from a set of candidate values in a given partition P_i, where each value belongs to a distinct attribute. The set of candidates includes only the values in the generalized record that represents the partition. Hence, a single specialization requires choosing a value from a maximum of d candidates, where d is the number of attributes in the raw dataset D. Selecting a value for specializing a partition is based on a real-valued score computed by a greedy utility function, which varies depending on the target workload. For example, the root partition in Figure 6.4 contains the candidate set $\{Any\_Continent, [1930-1999)\}$, where $[1930-1999)$ has been chosen for specialization.

Depending on the purpose of the data release, the chosen candidate should have either the highest or the lowest score. Scores give the sense of how much utility (or loss) the data has maintained (or suffered) due to specializing a candidate. For example if the released data is intended for classification analysis, then `Max` would be the utility function, and the "best" candidate for specialization is the one with the highest score.

Given a partition for specialization, *DiffMulti* computes the scores of the candidates in the partition. Then, *DiffMulti* makes use of the exponential mechanism, presented in Theorem 2.2, to select a candidate. The exponential mechanism exponentially favors higher scores while maintaining privacy. Scores are inverted for the

exponential mechanism in the cases where lower scores are more favorable. Herein, we present three utility functions; the *discernibility metric* and the *Normalized Certainty Penalty* are two general-purpose metrics for when the target workload of the released data is unknown, and the third utility metric is `Max` for a specific data analysis task, namely, classification analysis.

The first utility function is the *discernibility metric* (`DM`), which is a general-purpose quality metric that measures the amount of data loss the raw dataset has suffered due to anonymization [15]. Every record r in the anonymous dataset is assigned a penalty value equal to the size of the group of records that are indistinguishable from r. More specific to our method, a generalized record in a partition P_i is given a numeric penalty equal to $|P_i|$. Given a candidate g, $\texttt{DM}(D, g)$ is computed as follows:

$$\texttt{DM}(D, g) = \sum_{P_c \in child(P_g)} (|P_c|^2), \tag{6.1}$$

where P_g is g's partition, and $child(P_g)$ is the set of P_g's child partitions. Lower values of $\texttt{DM}(D, g)$ imply that specializing on g inflicts less anonymization distortion compared to other candidates. The discernibility penalty of the entire output dataset $\hat{D}$ can be computed as follows:

$$\texttt{DM}(\hat{D}) = \sum_{\forall P_i in \hat{D}} (|P_i|^2). \tag{6.2}$$

The sensitivity $\Delta\texttt{DM}(D, g) = 2|D| + 1$, where $|D|$ denotes the number of records in D. For simplicity, let us consider finding $\Delta\texttt{DM} = \max_{D,D'} ||\texttt{DM}(D) - \texttt{DM}(D')||_1$, where D and D' are two datasets differing by 1 record. Assume that the output dataset contains one partition. The square operation has a greater impact on a dataset D' s.t. $|D'| = |D| + 1$ (as opposed to $|D| - 1$). Thus, $\Delta\texttt{DM} = |D'|^2 - |D|^2 = 2|D| + 1$. The same reasoning can be applied to find $\Delta\texttt{DM}(D, g)$.

The second utility function is the *Normalized Certainty Penalty* (NCP), described in [135], which measures information loss due to generalization. Thus, lower values imply better data utility. Given a generalization hierarchy represented as a taxonomy tree $\mathbb{T}$, let e denote a node in the taxonomy tree and let $|\mathbb{T}|$ denote the number of leaf nodes in $\mathbb{T}$, where a leaf node represents a raw value. If a raw value a is generalized to node e_a in the taxonomy tree, then the NCP value of a is computed as follows:

$$\texttt{NCP}(a) = \begin{cases} 0, & e_a \text{ is leaf} \\ |e_a|/|\mathbb{T}|, & \text{otherwise,} \end{cases} \tag{6.3}$$

where $|e_a|$ is the number of leaf nodes of the subtree of the taxonomy tree rooted at e_a. For a numerical attribute A_{num}, $|e_a|$ would be the range of the generalization interval and $|\mathbb{T}|$ would be the domain range of A_{num}. Hence, $0 \leq \texttt{NCP}(i) \leq 1$. Given a candidate g, $\texttt{NCP}(D, g)$ is computed as follows:

$$\texttt{NCP}(D, g) = \sum_{c \in child(g)} (supp_c \cdot \texttt{NCP}(D, c)), \tag{6.4}$$

where $supp_c$ is the number of records that include a raw value generalized to c in the taxonomy tree. To find the sensitivity of $\texttt{NCP}(D, g)$ in Equation 6.4, let $\texttt{NCP}(D, c) = 1$. $supp_c$ can change at most by 1. Therefore, $\Delta\texttt{NCP}(D, g) = 1$. Lastly, the NCP of a generalized dataset $\hat{D}$ can be measured as follows:

$$\texttt{NCP}(\hat{D}) = \frac{\sum_{\forall i \in \hat{D}} (supp_i \cdot \texttt{NCP}(i))}{\sum_{\forall i \in \hat{D}} (supp_i)}. \tag{6.5}$$

Finally, our third utility function is Max, and it is used to publish an anonymous dataset for classification analysis. Given a candidate g, $\texttt{Max}(D, g)$ computes the score of g by summing the highest class frequencies of g's child values. In other words, if g's partition were to be specialized, the score of g would be equal to the result of adding the largest count in every child partition. $\texttt{Max}(D, g)$ helps to build a better classifier by going for the favor of choosing partitions with purest class frequencies.

Intuitively, higher scores are more desirable.

$$\texttt{Max}(D, g) = \sum_{c \in child(g)} (\max_{cls}(|D_c^{cls}|)). \tag{6.6}$$

The absence or addition of 1 record in D would change $\texttt{Max}(D, g)$ by at most 1. Therefore, the sensitivity of $\texttt{Max}(D, g)$ is 1.

Given a utility function u and a set of candidates $Cand_i$ from partition P_i, our algorithm utilizes the exponential mechanism to choose a candidate $g_i \in Cand_i$ with the following probability:

$$\frac{\texttt{exp}(\frac{\epsilon'}{2\Delta u} u(D, g_i))}{\sum_{g \in Cand_i} \texttt{exp}(\frac{\epsilon'}{2\Delta u} u(D, g))}, \tag{6.7}$$

where $u(D, g_i)$ is a score computed from any of the aforementioned utility functions, and Δu is the sensitivity of the utility function u. This step is outlined in Line 11 of Algorithm 6.1.

Theorem 6.1. *Choosing a candidate for specialization in accordance with Equation 6.7 satisfies ϵ'-differential privacy.* ■

In Chapter 6.2.2, we assumed that the input dataset D contains a categorical *Class* attribute, A^{cls}. In case A^{cls} is numerical, we need to make two changes. First, we replace the utility function `Max`, which is designed for a categorical *Class* attribute, with a score function that targets a numerical *Class* attribute. Second, in the evaluation process, we replace the classification model (e.g., *C4.5*) with a regression tree.

Allocating the Number of Specializations. One of the input parameters of Algorithm 6.1 is a positive integer that represents the maximum number of specializations, $h \in \mathbb{N}$. Specializing a single partition, i.e., distributing the records of the specialized partition among its newly created child partitions, decreases h by 1. The algorithm starts with one partition that includes the entire data records

generalized to the topmost level, followed by a series of specializations on each of the child partitions. This process gives rise to the following question: *what is the maximum number of specializations that can be performed on each child partition?* Differently put, how can the unused portion of h be distributed among the child partitions? For ease of presentation, let $P_i.h' \in \mathbb{N}$ refer to the maximum number of specializations that can be performed starting at partition P_i, where $0 \leq h' < h$. A straightforward solution would be to evenly distribute the remaining of h among the child partitions, as demonstrated by the following example.

Example 6.3.2. Let $h = 21$. Suppose partition P_1 is specialized and two child partitions, P_2 and P_3, are created. Initially, $P_i.h' = h = 21$. $P_1 \rightarrow \{P_2, P_3\}$ consumes 1 specialization from h. As a result, $P_2.h' = P_3.h' = (21 - 1)/2 = 10$. Therefore, every child partition gets to be iteratively specialized 10 times at most. ■

The above method is simple and does not violate differential privacy because the sensitivity of evenly distributing $h' - 1$ among child partitions is 0. Having said that, recall that every partition represents a region in the multidimensional space of the input dataset D. It is unlikely that the data points (raw records) are evenly distributed throughout the domain space. Indeed, some regions are denser than others.

Example 6.3.3. In Example 6.3.2, we showed that if the remainder of h was evenly distributed among the child partitions, then $P_2.h' = P_3.h' = 10$. Assume that $|P_1| = 10$, $|P_2| = 8$, and $|P_3| = 2$, where $|P_i|$ is the number of records in P_i. This means that a set of 2 records and a set of 8 records will be specialized equally. ■

Even distribution of the number of specializations does not take into consideration dense regions in the domain space. Shifting specializations towards dense regions has a positive impact on data utility because more records get specialized,

hence producing less abstract data. We make a contribution in this work by imposing a "fair" distribution of the number of specializations rather than an even one, which causes more data distortion. Let $P_i \rightarrow \{P_{c_1}, P_{c_2}, \ldots, P_{c_v}\}$, the number of specializations assigned to a child partition P_{c_i} is proportional to its size. More specifically,

$$P_{c_i}.h' = \begin{cases} 0, & |P_{c_i}| = 0 \\ \lfloor \frac{|P_{c_i}|}{|P_i|} \cdot (P_i.h' - 1) \rfloor, & \text{otherwise,} \end{cases} \tag{6.8}$$

where $P_{c_i}.h' \in \mathbb{N}$. Applying the new distribution mechanism to Example 6.3.2, we get $P_2.h' = (8/10) \cdot (21 - 1) = 16$ and $P_3.h' = (2/10) \cdot (21 - 1) = 4$.

Equation 6.8 does not respect the indeterministic nature of differential privacy. This is because, given a partition P_x, Equation 6.8 calculates the exact record count in that partition, $|P_x|$, which may change for a neighboring input dataset D'. Therefore, we use the Laplace mechanism (Theorem 2.1) to return noisy record counts, instead. Given 1 as the sensitivity of a count query, a privacy budget ϵ', and a true record count $|P_x|$, a differentially-private version of Equation 6.8 calculates $|P_x| + \texttt{Lap}(1/\epsilon')$. This step is outlined in Line 13 of Algorithm 6.1.

Theorem 6.2. *Fair distribution of the number of specializations satisfies ϵ'-differential privacy if Equation 6.8 adds $\texttt{Lap}(1/\epsilon')$ to true record counts.* ■

If the calculated value of $P_{c_i}.h'$ contains a fraction, then the value is rounded down to the nearest non-negative integer. If the summation of all fractions among the child partitions is ≥ 1, then this remainder can be randomly distributed among the child partitions to avoid losing portions of h. Note that this post-processing step conforms with differential privacy because this step is a public rule and is not dependent on the input dataset [73].

6.3.3 Determining a Numerical Split Point

Given a partition and its set of candidates, where each candidate has a real-valued score, a specialization is performed on the value chosen by the exponential mechanism. We refer to this value as the *split value*. A specialization $g \rightarrow child(g)$ replaces the split value g with its child values. We perform a specialization as follows: in a partition P_i, once a value $g \in P_i$ is chosen for specialization, our algorithm creates a child partition P_{c_j} for every child value $c_j \in child(g)$. Then, it distributes the records in P_i among the child partitions where each record goes to its pertinent child partition. A score is computed for the new values in every child partition. Finally, the parent partition P_i is deleted as it contains no further useful information to the specialization process. We note that the raw values of the records are maintained throughout runtime in order to properly split the records among child partitions.

To satisfy differential privacy, any operation should be independent of the underlying data. We project this rule on finding the split value for both categorical and numerical attributes. We note that a *categorical split value* is a generalized value drawn from a pre-defined generalization hierarchy (taxonomy tree), and a *numerical split value* is an interval over the continuous domain. If the exponential mechanism chooses a categorical split value, then adding or removing a record from the input dataset will not affect how a split value is determined. As a result, having a taxonomy tree induces sensitivity $= 0$ on determining a categorical split value. On the other hand, if the exponential mechanism chooses a numerical split value, then a numerical *split point* has to be found from the chosen interval. Due to the inconvenience of defining a taxonomy tree for numerical attributes, our proposed algorithm adaptively determines a numerical split point that divides an interval over the attribute domain into two disjoint and successive subintervals. A simple solution to finding a split point can be to pick a random point from the raw records and split the records into two child partition accordingly. This solution suffers from two shortcomings. First, the chosen point may not exist in a neighboring dataset

that differs by 1 record. Hence, choosing a point directly from the raw data records violates differential privacy. Second, this solution does not pay attention to the utility of the partitioned data. Next, we present a method for carefully choosing a numerical split point that maximizes data utility without violating differential privacy.

We compute a score for every value g_i in the domain of the numerical attribute $\Omega(A_{num})$, then use the exponential mechanism to choose a value $g_i \in \Omega(A_{num})$ with the following probability:

$$\frac{\texttt{exp}(\frac{\epsilon'}{2\Delta u}u(D, g_i))}{\int_{g\in\Omega(A_{num})} \texttt{exp}(\frac{\epsilon'}{2\Delta u}u(D, g))\, dv}, \tag{6.9}$$

where $u(D, g_i)$ is a score computed from a utility function u with sensitivity Δu.

Computing a score for every value in the domain can be exhaustive. We observe that intervals of consecutive values along the attribute domain can have the same score. More formally,

Observation 6.1. Given a utility function u and a numerical attribute A_{num} with domain range $\Omega(A_{num}) = \{I_1, I_2, \ldots, I_v\}$, where I_i is an interval of consecutive values. Then, $\forall I_i \in \{I_1, I_2, \ldots, I_v\}$, we have $u(D, g_i) = u(D, g_j)\ \forall g_i, g_j \in \Omega(I_i)$. ■

Following this observation, we partition the domain range into successive intervals $\{I_1, I_2, \ldots, I_v\}$, where every two successive intervals are separated by a numerical value from the input data. We generated a score for every interval and use the exponential mechanism to choose an interval with the following probability:

$$\frac{\texttt{exp}(\frac{\epsilon'}{2\Delta u}u(D, g_i)) \times |\Omega(I_i)|}{\sum_{j=1}^{v}(\texttt{exp}(\frac{\epsilon'}{2\Delta u}u(D, g_j)) \times |\Omega(I_j)|)}, \tag{6.10}$$

where $|\Omega(I_i)|$ is the size of the interval. Once the exponential mechanism returns an interval, we uniformly sample a value from the returned interval, since all its values have the same score.

Example 6.3.4. Let us reexamine Figure 6.4. The numerical split value $[1930 - 1999)$ is chosen for specialization at the root node. Therefore, we partition the domain of attribute YoB into the set $\{[1930-1947), [1947-1953), [1953-1955), [1955-1957), [1957-1959), [1959-1968), [1968-1999)\}$. After that, a score is generated for each interval. Let us assume that interval $[1959 - 1968)$ has the highest score among all the other candidate intervals in the set. Finally, a split point is randomly picked from $[1959 - 1968)$, which is 1960 in this example. ■

As we mentioned in Chapter 6.3.2, the `NCP` of an interval I_i is computed from the ratio $\frac{|\Omega(I_i)|}{|\Omega(A_{num})|}$. This results in every value in the interval having a slightly different score than the other. To avoid finding a score for every single value, we choose the midpoint to represent the score and split point of an interval. The step of determining a numerical split point is outlined in Line 9 of Algorithm 6.1.

Theorem 6.3. *Choosing a numerical split point in accordance with Equation 6.10 satisfies ϵ'-differential privacy.* ■

6.3.4 Publishing Record Counts

In Chapter 6.3.1, we mentioned that *DiffMulti* is comprised of two phases: iteratively specializing records, followed by publishing generalized records. Phase 1 ends when no further specializations can be performed. Phase 2 publishes record counts in every leaf partition that resulted from Phase 1. However, as discussed earlier in Chapter 6.3.2, publishing true counts violates differential privacy because counts change depending on the underlying dataset. Again, we use the Laplace mechanism to publish a noisy version of the true count. Figure 6.4 shows the added Laplacian noise in a dashed ellipse in every leaf partition.

Recall from Chapter 6.3.1 that a partition holds $|\Omega(A^{cls})|$ groups of generalized records, we wish to release noisy group counts that satisfy $\frac{\epsilon}{2}$-differential privacy. Therefore, in Line 21, Algorithm 6.1 publishes $Count + \texttt{Lap}(2/\epsilon)$ for every $P_i \in \mathcal{P}_L$,

where *Count* is the true group count in a leaf partition and $\mathcal{P}_L$ denotes the set of leaf partitions. Noisy record counts are rounded down to the nearest non-negative integer, a step that does not violate differential privacy [73].

Theorem 6.4. *Publishing record counts satisfies* $\frac{\epsilon}{2}$*-differential privacy.* ■

Next, we will examine the properties of differential privacy to prove that *DiffMulti* is differentially-private.

Privacy Budget Allocation. To satisfy differential privacy, it is imperative for any operation to be insensitive to the underlying dataset. In Chapters 6.3.2-6.3.4, we investigated the main operations of *DiffMulti* and showed that each operation on its own is differentially-private. To prove that *DiffMulti* is differentially-private as a whole, we will examine sequential composition (Lemma 2.1) and parallel composition (Lemma 2.2) in order to carefully allocate a privacy budget $0 < \epsilon' \leq \epsilon$ to each operation, where ϵ is an input parameter.

The algorithm exhibits two behaviors in which parallel composition can be witnessed. In Phase 1, specializing a partition $P_i \rightarrow \{P_{c_1}, \ldots, P_{c_v}\}$ results in child partitions containing disjoint sets of records. Hence, allocating a privacy budget $= \epsilon_x$ to each child partition $P_{c_j} \in \{P_{c_1}, \ldots, P_{c_v}\}$ will result in a total budget consumption $= \epsilon_x$ among $\{P_{c_1}, \ldots, P_{c_v}\}$. In Phase 2, a leaf partition contains $|\Omega(A^{cls})|$ groups of disjoint records. Again, assigning ϵ_x to each group in the partition to publish noisy counts will consume a total of ϵ_x.

Sequential composition manifests along a root-to-leaf path in the tree of partitions because the same set of records is being iteratively processed, starting from the topmost-general root partition until a leaf partition is reached. Moreover, the set of root-to-leaf paths falls under parallel composition, as we explained above. Therefore, the privacy budget consumption is added along the *longest* root-to-leaf path.

Let us start by examining the amount of privacy budget needed to process a

single partition. Lines 9, 11, and 13 of Algorithm 6.1 outline three ϵ'-differentially-private operations. Therefore, a single partition requires $3 \times \epsilon'$ to complete a specialization in a differentially-private way. Line 9 is a special case for the topmost-general root partition because we need to find a split point for *all* numerical attributes, whereas any subsequent child partition contains *at most one* numerical value for which we need to find a split point. Hence, Line 9 computes a split point $|A_{num}|$ times in the root partition, where $|A_{num}|$ is the number of numerical attributes. Given ϵ as the entire privacy budget for Algorithm 6.1, we dedicate $\frac{\epsilon}{2}$ to specializing partitions along a root-to-leaf path (Phase 1) and $\frac{\epsilon}{2}$ for publishing record counts (Phase 2). The Laplacian noise in Phase 2 is, therefore, $\texttt{Lap}(2/\epsilon)$. The budget for each of the three differentially-private operations in Phase 1 $\epsilon' = \frac{\epsilon}{2(|A_{num}|+3|\mathfrak{n}|)}$, where $|\mathfrak{n}|$ is the length of the longest path.

In conclusion, based on the above privacy analysis and Theorems 6.1- 6.4, *DiffMulti* is ϵ-differentially private.

6.3.5 Proposed Algorithm

Algorithm 6.1 outlines the key steps of our proposed method, *DiffMulti*. Lines 1-20 describe Phase 1, and Line 21 describes Phase 2. The algorithm accepts three input parameters: a raw dataset D, a privacy budget ϵ, and the maximum number of specializations h. The output is an ϵ-differentially-private dataset $\hat{D}$.

In Phase 1, *DiffMulti* performs a sequence of specializations on partitions. A specialization is performed only if it is *valid*. A partition that does not incur a valid specialization is considered a leaf partition, which is used in Phase 2 to publish noisy record counts.

Definition 6.3 (*A valid specialization*)**.** Given a partition P_i, we say that P_i incurs a *valid specialization* (or a specialization on P_i is *valid*) iff all the following hold true: (1) the number of specializations h' assigned to P_i, $P_i.h' > 0$, (2) $\exists g \in P_i$

Algorithm 6.1: *DiffMulti*

Input: Raw dataset D, privacy budget ϵ, and number of specializations h
Output: Diff-private & multidim-generalized dataset $\hat{D}$

1: $P_i \leftarrow$ Initialize every value in D to the topmost value;
2: $\mathcal{P}_T \leftarrow P_i$; // Temporary partitions
3: $\mathcal{P}_L \leftarrow \emptyset$; // Leaf partitions
4: $|\mathfrak{n}| = \sum^{|\mathcal{A}|} height\ of\ \mathbb{T}_i$;
5: $\epsilon' \leftarrow \frac{\epsilon}{2(|A_{num}|+3|\mathfrak{n}|)}$;
6: **while** $\mathcal{P}_T$!= $\emptyset$ **do**
7: $P_i \leftarrow \mathcal{P}_T$.pop();
8: **if** P_i incurs a valid specialization **then**
9: Determine a split point for every new valid $g_{num} \in P_i$ with probability $\propto \texttt{exp}(\frac{\epsilon'}{2\Delta u} u(D, g_{num}))$;
10: Compute the score for every valid $g \in P_i$;
11: Select $g \in P_i$ with probability $\propto \texttt{exp}(\frac{\epsilon'}{2\Delta u} u(D, g))$;
12: Specialize $P_i \rightarrow \{P_{c_1}, \ldots, P_{c_v}\}$;
13: Use Equation 6.8 to determine, with $\texttt{Lap}(1/\epsilon')$, the number of specializations h' assigned to every $P_{c_j} \in \{P_{c_1}, \ldots, P_{c_v}\}$;
14: $\mathcal{P}_T \leftarrow \mathcal{P}_T - P_i$;
15: $\mathcal{P}_T$.push(P_{c_j}) for every $P_{c_j} \in \{P_{c_1}, \ldots, P_{c_v}\}$;
16: **else**
17: $\mathcal{P}_T \leftarrow \mathcal{P}_T - P_i$;
18: $\mathcal{P}_L \leftarrow \mathcal{P}_L \cup P_i$;
19: **end if**
20: **end while**
21: **return** each group in $P_i \in \mathcal{P}_L$ with count $(Count + \texttt{Lap}(2/\epsilon))$

s.t. $|child(g)| \neq 0$, and (3) the number of specializations from the root partition to $P_i \leq |\mathfrak{n}|$. ■

Line 1 generalizes all raw values in D to one root partition, P_i, that contains the topmost values. Let $\mathcal{P}_T$ be a list that structures data as a LIFO stack. Line 2 initializes the list of *temporary partitions* $\mathcal{P}_T$ with P_i. At any given iteration, $\mathcal{P}_T$ holds the partitions to be later specialized. Line 3 initializes the set of *leaf partitions* $\mathcal{P}_L$. Line 4 sets the length of the longest root-to-leaf path to the total number of levels of all the taxonomy trees $\{\mathbb{T}_1, \ldots, \mathbb{T}_{|\mathcal{A}|},\}$ that belong to all attributes $\mathcal{A}$ in D except the *Class* attribute. We further elaborate on this point in Chapter 6.3.6.

Line 5 is executed once to initialize ϵ', as per the discussion in Chapter 6.3.4. This ϵ' will be used for all the differentially-private operations in the algorithm. Lines 6-20 recursively perform specializations on the partitions in $\mathcal{P}_T$. Line 7 provides the next partition to be specialized, P_i, and Line 8 checks its validity. If P_i does not yield a valid specialization, it is removed from $\mathcal{P}_T$ and added to $\mathcal{P}_L$ in Lines 17 and 18. Otherwise, Line 9 chooses, in a differentially-private way, a split point for every new numerical value $g_{num} \in P_i$. This step creates two child values for g_{num} with non-overlapping intervals. Line 10 computes a score for every valid value $g \in P_i$. A utility function is chosen beforehand depending on the target workload. Line 11 uses the exponential mechanism to select a value g. Line 12 specializes $g \rightarrow child(g)$ and creates a partition P_c for every child value $c \in child(g)$. After that, Line 13 assigns a noisy number of specializations h' to each child partition P_c in proportion to the number of records in P_c, where $0 \leq h' < h$. Line 14 removes the parent partition P_i from $\mathcal{P}_T$, whereas Line 15 pushes the set of child partitions $\{P_{c_1}, \ldots, P_{c_v}\}$ onto the beginning of $\mathcal{P}_T$. The recursion (Lines 6-20) stops when the list of temporary partitions $\mathcal{P}_T$ becomes empty, i.e., no further valid specialization can be performed on any partition $\in \mathcal{P}_T$. The algorithm terminates when Line 21 synthesizes the ϵ-differentially-private dataset $\hat{D}$ by returning a Laplacian-noisy version of the counts in all the partitions in $\mathcal{P}_L$.

Top-Down Specialization. Algorithm 6.1 uses a tree data structure (Figure 6.4) reminiscent of the one proposed in [45]. A significant difference between both relies in the fact that our structure is not indexed. Rather, our technique works on the node level due to the multidimensional aspect of our algorithm, where every node in the tree represents a partition containing a unique set of records. Such structuring of data records to generate a differentially-private and multidimensionally-generalized version of a raw dataset provides the following advantages. First, the exponential mechanism chooses from the set of candidates that exists in the same

partition. This avoids having to link other candidates by scanning other partitions. Second, in any iteration, all records being specialized exist within the same partition, which provides direct access to those records. This provides efficiency by avoiding scanning the entire dataset when computing scores or splitting records from parent to child partitions. Third, the tree structure allows Algorithm 6.1 to organize child partitions in a LIFO stack so that the tree grows in a depth-first order. This provides space efficiency by performing a sequence of specializations on records that exist within the same memory block, i.e., moving from parent to child, as opposed to moving from parent to sibling. Fourth, Algorithm 6.1 does not need to complete a full run to finish anonymizing the data; rather, the data is anonymous at the end of any iteration, and a noisy version is ready to be published.

Complexity Analysis. We find the complexity of *DiffMulti* in terms of the number of records in the input dataset, $|D|$. Algorithm 6.1 starts with a root partition that contains all data records generalized to the topmost level over all attributes (Line 1). Then, a split point has to be found for *all* numerical attributes (Line 9). It is important to note that this step is performed at the root partition solely as subsequent partitions will contain *at most one* new numerical candidate for which a split point is to be found. Recall that every partition contains a set of records generalized to the same hierarchical level. Given a numerical attribute A_{num}, a split point is found by first sorting the partition records on A_{num}, and then scanning them once to compute a score for every numerical value on A_{num} that appears in the partition records. Sorting partition records on a single numerical attribute costs $O(|P_i| \log |P_i|)$, where $|P_i|$ is the number of records in P_i. If P_i is the root partition, then $|P_i| = |D|$; otherwise, $|P_i|$ is usually much smaller than $|D|$. Thus, Line 9 costs $O(|A_{num}| \times |D| \log |D|)$ for the root partition and $O(|P_i| \log |P_i|)$ otherwise, where $|A_{num}|$ is the number of numerical attributes.

In Line 10, a score is computed for every candidate in the current partition.

Again, the root partition is a special case in which data records are scanned once for every attribute to determine the score. For all other partitions, no record scanning is necessary because all the required pieces of information have been obtained in an earlier iteration (Line 12); thus, this operation can be done in a constant time. Therefore, given a set of attributes $\mathcal{A}$, and assuming that all candidates in P_i are valid, Line 10 costs $O(|\mathcal{A}| \times |P_i|)$ for the root partition and $O(1)$ otherwise.

After computing all scores, the exponential mechanism in Line 11 returns a split value. The complexity of the exponential mechanism is proportional to the number of selections from which the mechanism will choose. The exponential mechanism appears twice in the algorithm. First, in Line 9 the purpose is to select a split point from a given numerical attribute. This process requires partitioning the numerical attribute into successive intervals $\{I_1, I_2, \ldots, I_v\}$ then choosing one interval from the set (Example 6.3.4). Therefore, the cost of Line 9 is $O(v)$. Second, in Line 11 the purpose is to select a candidate from a set of candidates, which has a maximum size equal to the number of attributes in D. Hence, the cost of Line 11 is $O(|\mathcal{A}|)$. $|D|$ is usually much larger than the number of intervals and the number of attributes. We, therefore, neglect the complexity of the exponential mechanism.

In Line 12, the current partition is specialized on the selected value g from Line 11. The records in P_i are distributed among its child partitions, a step that requires scanning those records to determine the child partition to which every record belongs. Thanks to the tree structure, partition P_i gives direct access to pertinent records in D. To boost the efficiency of our implementation, along with record scanning, we collect pieces of information that will be used in computing the scores in the child partitions in subsequent iterations (Line 10). Specifically, for each $c \in child(g)$ we collect $|D_c|$, $|D_c^{cls}|$, $|D_z|$, and $|D_z^{cls}|$, where cls is a value on the *Class* attribute and $z \in child(c)$. Thus, specializing a partition costs $O(|P_i|)$. The rest of the lines in Algorithm 6.1 are done in a constant time.

The most expensive operation in the algorithm is sorting partition records

(Line 9). Given that the algorithm performs at most h specializations, and given a fixed number of attributes, the total cost of the algorithm is $O(h \times |P_i| \times log|P_i|)$. In order to express the time complexity in terms of the number of records in the input dataset, $|D|$, we assume the worst case by which partitions are specialized according to a *binary* tree structure where every leaf partition contains 1 record. Every level collectively processes $|D|$ records and the tree can have at most $log|D|$ level. As a result, *DiffMulti* has a worst-case runtime of $O(|D| \cdot \log |D| \cdot \log |D|)$.

6.3.6 Discussion

In Chapter 6.3.4, we discussed privacy budget allocation and how to compute ϵ' given that partitions are structured as a tree. In such tree, the *length* of a root-to-leaf path refers to the number of specializations along that path. Thanks to the parallel composition property, we only need to consider the length of the longest root-to-leaf path, $|\mathfrak{n}|$, when computing ϵ' because $\mathfrak{n}$ performs the longest sequence of differentially-private operations. For example, the length of the longest root-to-leaf path in Figure 6.4 is 3.

Knowing $|\mathfrak{n}|$ in advance is a challenging task because: (a) the depth of the tree of partitions is indeterministic due to multidimensional generalization; and (b) ϵ' needs to be computed in advance so that *DiffMulti* can perform the differentially-private operations in Phase 1 (Lines 9, 11, and 13). Herein, we provide theoretical and practical estimations of $|\mathfrak{n}|$.

Every partition specialization in the tree decreases the number of specializations h by 1. We consider the following two cases of tree growth. If the tree fans out, i.e., the upper levels contain a large number of partitions, then h will be mainly consumed horizontally. With little h left, the tree will not grow deeper. However, if the tree branches with small number of child partitions, then more h will be available to allow for the tree to grow deeper. Our interest lies in the case where the tree grows as deep as possible; therefore, we consider a perfect binary tree of

partitions where all leaf partitions are at the same level and every leaf partition contains 1 record from D. With $|D|$ leaf partitions, the hight of the tree would be $log_2|D|$. Hence, the length of the longest path is $|\mathfrak{n}| = log_2|D|$. We note that this is a theoretical estimate built on the assumption that all data records conform to the perfect binary distribution of partitions. It is safe to say that real-life datasets are unlikely to follow such tree structure as some child partitions are likely to be empty causing the tree to grow deeper than $log_2|D|$ levels. An alternative, and more practical, solution is to consider the number of specializations required to transform a record from its topmost general version to its raw version. This is achieved by summing the heights of all the taxonomy trees for a given dataset. The result is an integer that represents the length of longest possible root-to-leaf path. For all the experiments in the following section, we use the latter approach to estimate $|\mathfrak{n}|$.

6.4 Experimental Evaluation

We evaluate the performance of our proposed method, *DiffMulti*, with respect to the utility of the output data, efficiency in terms of runtime, and scalability for handling large datasets. The goal is to measure the impact of multidimensional generalization achieved by means of differential privacy. We compare our proposed algorithm with 3 closely related ones: (1) *DiffGen* [96], a differentially-private algorithm that performs single-dimensional (global) generalization; (2) *Mondrian* [76,78], an algorithm that enforces multidimensional generalization to publish k-anonymous data; and (3) *PrivBayes* [141], a differentially-private algorithm that utilizes Bayesian networks to release high-dimensional data that approximate the distribution of the input data. Furthermore, we report the data utility from our method and *RPS* [106], a general framework for releasing relational data under differential privacy.

For all our evaluations, we use the widely employed and publicly available *Adult* dataset [40]. *Adult* is a census dataset that contains 45,222 records and 15 attributes

comprising 8 categorical, 6 numerical, and 1 *Class* attribute that represents two income values, "$\leq$ 50K" and "$>$ 50K".

When comparing *DiffMulti* with *DiffGen*, we vary the number of specializations h in *both* methods and report the results. h is an integer and its maximum limit is tied to the taxonomy trees. *DiffGen* employs single-dimensional generalization, and thus h can be at most equal to the total of the number of non-leaf nodes in all the taxonomy trees, whereas *DiffMulti* employs multidimensional generalization resulting in h being massively greater than *DiffGen*'s h. More specifically, the maximum number of specializations for *DiffMulti* is computed as follows:

$$\begin{aligned} h = 1 &+ Inter_1 \\ &+ Leaves_1 + (Leaves_1 \times Inter_2) \\ &+ (Leaves_1 \times Leaves_2) + (Leaves_1 \times Leaves_2 \times Inter_3) \\ &+ \ldots \\ &+ (Leaves_1 \times Leaves_2 \times \ldots \times Leaves_{w-1}) \\ &+ (Leaves_1 \times Leaves_2 \times \ldots \times Leaves_{w-1} \times Inter_w), \end{aligned} \tag{6.11}$$

which can be rearranged as follows:

$$\begin{aligned} h = 1 &+ Inter_1 \\ &+ Leaves_1(1 + Inter_2) \\ &+ (Leaves_1 \times Leaves_2)(1 + Inter_3) \\ &+ \ldots \\ &+ (Leaves_1 \times Leaves_2 \times \ldots \times Leaves_{w-1})(1 + Inter_w). \end{aligned} \tag{6.12}$$

Thus, the maximum number of specializations for *DiffMulti* is:

$$h = 1 + Inter_1 + \sum_{i=1}^{w-1}[(\prod_{j=1}^{i} Leaves_j)(1 + Inter_{i+1})], \tag{6.13}$$

where $Inter_i$ is the number of intermediate nodes in taxonomy tree i, $Leaves_i$ is the number of leaf nodes in taxonomy tree i, and w is the total number of taxonomy trees. Considering only the taxonomy trees of the categorical attributes of the *Adult* dataset, h has a limit of nearly 50 for *DiffGen*, while for *DiffMulti* h is in millions. Those numbers assume that every combination of raw values exists in the raw datasets, which is not the case empirically. We reasonably vary the value of h in order to maintain consistent results for both methods.

The settings and configurations of the experiments are as follows: We implemented our method in C++. All experiments run on a PC with an Intel Core i5 CPU, 2.4GHz, and 8GB of RAM. Unless otherwise mentioned, we run each experiment five times and report the average of the obtained results in the graphs herein. Moreover, recall from Chapter 6.3.4 that $\epsilon' = \frac{\epsilon}{2(|A_{num}|+3|\mathfrak{n}|)}$, where $\mathfrak{n}$ is the longest root-to-leaf path. Chapter 6.3.6 presented an estimation of $|\mathfrak{n}|$ by summing the heights of all the taxonomy trees. *Adult* has 8 distinct taxonomy trees pertaining to 8 categorical attributes. Summing the heights of these taxonomy trees yields 21. For the remaining 6 numerical attributes, no such hierarchy exists and so we assume the height of each attribute's taxonomy tree is 7 on average. Hence, the total sum is 63, and thus we set $|\mathfrak{n}| = 63$ for all our experiments. Specialization stops when a root-to-leaf path reaches length $= |\mathfrak{n}|$ as the dedicated privacy budget would have been consumed entirely. If the longest root-to-leaf path stopped before reaching $|\mathfrak{n}|$, the unused portion of the privacy budget is added to the privacy budget dedicated to the leaf partitions in order to preserve the entire budget ϵ. It is worth noting that choosing a height for a numerical taxonomy tree is a loose assumption, and reasonably varying this number has no significant impact on *DiffMulti*.

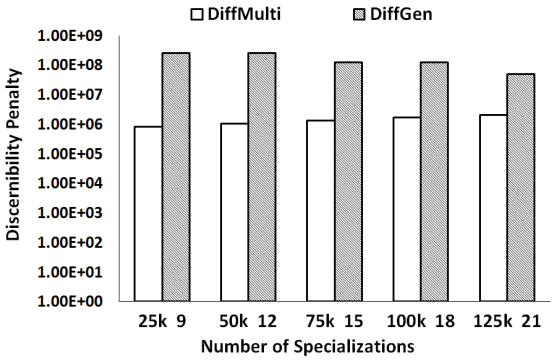


Figure 6.5: Comparing *DiffMulti* and *DiffGen* in terms of discernibility penalty

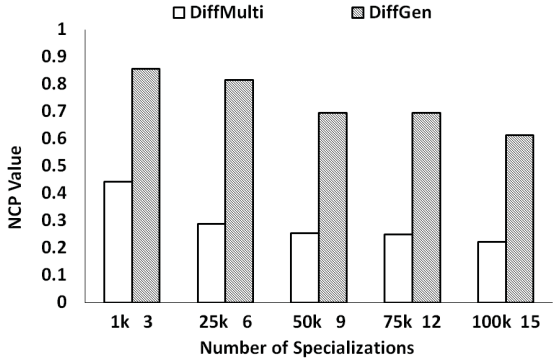


Figure 6.6: Comparing *DiffMulti* and *DiffGen* in terms of NCP

6.4.1 Data Utility

Figures 6.5 and 6.6 depict the utility of the output dataset where discernibility [15] and NCP [135] are the utility functions, respectively. Recall from Chapter 6.3.2 that both functions are general-purpose metrics that measure the amount of distortion in the output dataset with comparison to its raw version. We compare our method *DiffMulti* with *DiffGen* in order to showcase the impact of multidimensional generalization over single-dimensional. In both figures, we vary the number of specializations of both methods and measure the distortion; higher reported values indicate higher distortion. Figure 6.5 suggests that *DiffMulti* is able to significantly reduce the discernibility penalty by at least one order of magnitude than *DiffGen*. Figure 6.6 reports the NCP values, where $0 \leq \texttt{NCP} \leq 1$ and $\texttt{NCP} = 1$ means that the

output dataset is generalized to the topmost value in the taxonomy tree of every attribute. Again, *DiffMulti* manages to maintain significantly less generalized output data thanks to its multidimensional approach. Both figures suggest that *DiffMulti* is robust with respect to the number of specializations as the incurred distortion does not change significantly.

We also test our method on a specific target workload, i.e., classification accuracy. The objective is to build a classification model that accurately predicts the *Class* attribute in *Adult*; particularly, if an individual has income "> 50K". To achieve this goal, first we use the training records to create generalization regions in the multidimensional space. Then, we generalize the training and testing records to those regions. Training records are used to build a classification model, which in turn is used to predict the *Class* attribute in the testing records. For this set of experiments, we used two widely deployed classifiers: *C4.5* [109] and *SVM* [70]. *C4.5* is trained with 2/3 of *Adult* records and tested with the remaining 1/3 records, whereas *SVM* is trained with 4/5 of the records and tested with the remaining 1/5. For both classifiers, we set `Max` to be the utility function.

Before applying any anonymization method on *Adult*, we first classify *Adult* as a raw dataset to measure the *Baseline Accuracy* (BA), which is the best achieved accuracy. We aspire to obtain results close to the BA when classifying the output data of an anonymization method, which typically results in lower accuracies than the BA. Moreover, we measure the *Lower Bound Accuracy* (LA), which considers solely the *Class* attribute when training and testing the classifier. LA gives the sense of having a worst-case accuracy that should be surpassed by the output of a proposed anonymization method. Figures 6.7, 6.8, and 6.9 measure classification accuracy in percentage where higher values imply better accuracy.

Figure 6.7 depicts the impact of differential privacy on classification accuracy using *C4.5* classifier. We test *DiffMulti* by varying the privacy budget $0.1 \leq \epsilon \leq 1$ and the number of specializations $300 \leq h \leq 1,500$. The privacy budget should not

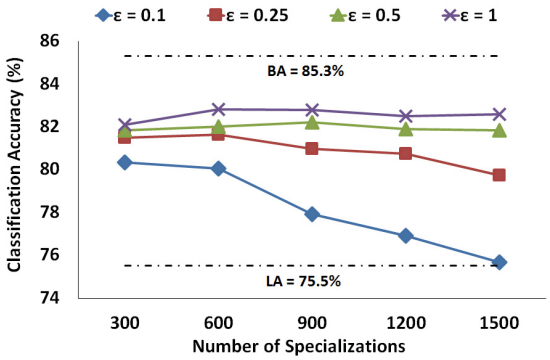


Figure 6.7: *DiffMulti*: the impact of ϵ on classification accuracy

be set large, i.e., typically ≤ 1 [34, 36, 41]. Higher privacy budgets result in more accurate classification because the differentially-private operations of the algorithm, i.e., data partitioning and publishing record counts, incur less noisy outputs.

Figure 6.8 compares *DiffMulti* with both *DiffGen* and *Mondrian* in terms of classification accuracy using *C4.5* classifier. We set $\epsilon = 1$ in both *DiffMulti* and *DiffGen* and vary the number of specializations $1,000 \leq h \leq 10,000$ and $2 \leq h \leq 20$, respectively. For *Mondrian*, we set $k = 60$. We observe that *DiffMulti* is dominant in achieving higher accuracy, especially at lower numbers of specializations. However, for higher values of h, *Mondrian* achieves up to 2% better accuracy than *DiffMulti*. We note that the 2% improvement in accuracy when applying k-anonymity-based *Mondrian* comes at the cost of releasing anonymous data that is vulnerable to syntactic-based privacy attacks [126] [48] [72].

The authors of *DiffGen* conducted a similar experiment in [96] to compare the classification accuracy of their non-interactive *DiffGen* with the interactive *DiffP-C4.5* [41], an algorithm that maintains differential privacy when building a classifier. Although *DiffP-C4.5* achieved promising results, *DiffGen* obtained improved accuracy at $\epsilon = 1$. However, the latter was constantly surpassed by *DiffMulti*, as suggested by Figure 6.8.

Figures 6.7 and 6.8 show that as the number of specializations increases, the

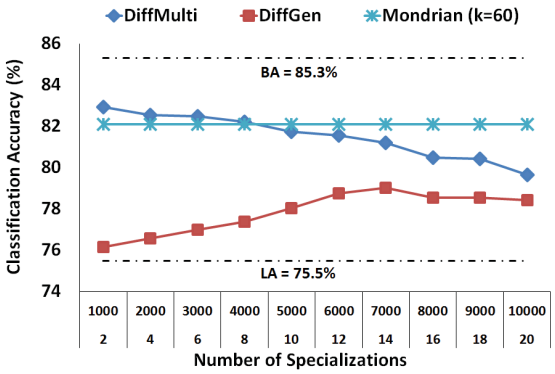


Figure 6.8: Comparing *DiffMulti*, *DiffGen*, and *Mondrian* in terms of classification accuracy

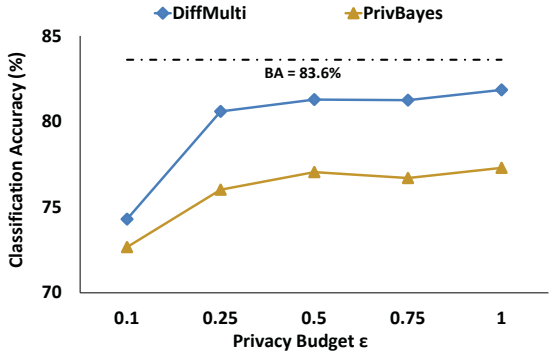


Figure 6.9: Comparing *DiffMulti* and *PrivBayes* in terms of classification accuracy

classification accuracy of *DiffMulti* decreases. This is due to the following two reasons. First, a higher number of specializations results in longer root-to-leaf paths. Consequently, the computed ϵ' assigned to a single differentially-private operation becomes very small compared to that of *DiffGen*, resulting in poor partitioning decisions that contribute to building an inaccurate classifier. Second, a higher number of specializations yields a larger number of leaf partitions. Consequently, each partition contains fewer records. As the number of records decreases, the effect of the added Laplace noise in Phase 2 of Algorithm 6.1 increases. With a relatively large noise, true record counts undergo severe distortion, which in turn adversely affects building a proper decision tree. This finding confirms that for high-dimensional

data, publishing noisy counts of every possible combination of domain values greatly degrades the quality of analysis.

Figure 6.9 depicts the performance of *DiffMulti* and *PrivBayes* using *SVM* classifier. We set the number of specializations for *DiffMulti* $h = 10,000$ and used the default parameters for *PrivBayes*. We ran both methods over 5 different privacy budgets $0.1 \leq \epsilon \leq 1$ and reported the resulting accuracy of each run. *DiffMulti* constantly performs more accurate predictions than *PrivBayes* even at low privacy budgets (noisy output data). We note that, unlike *PrivBayes* which outputs raw data values, our solution performs a series of carefully selected generalizations which contribute to improving classification accuracy, as suggested by Figure 6.8.

We carry out an additional experiment in which we compare *DiffMulti* with the *RPS* framework [106] in terms of classification accuracy. The accuracy of *RPS* was taken from the authors' paper. The same settings of *RPS* experiments were applied when conducting our experiment: *Adult* dataset of 30,162 records and 11 attributes (See [106]), and 5-fold cross validation for measuring the classification accuracy. We fixed $\epsilon = 1$ for both methods and obtained the following results. The raw data $BA = 82.96\%$. For *RPS*, 79% was the highest achieved accuracy, whereas *DiffMulti* achieved 81.54% when the number of specializations was set to $1,000$. While our method is able to achieve comparable classification accuracy to *RPS*, the latter does not scale for high-dimensional datasets [108].

6.4.2 Efficiency

In this experiment, the objective is to measure the runtime of our method and compare the results to those achieved by *DiffGen*. Figure 6.10 depicts the runtime in seconds where the utility function is `Max`, $\epsilon = 1$, and the number of specializations is $10,000 \leq h \leq 100,000$ for *DiffMulti* and $2 \leq h \leq 20$ *DiffGen*. The runtime of *DiffMulti* when performing $100,000$ specializations is nearly 30 seconds. Results suggest that our method, though it runs a few seconds longer than *DiffGen*, is

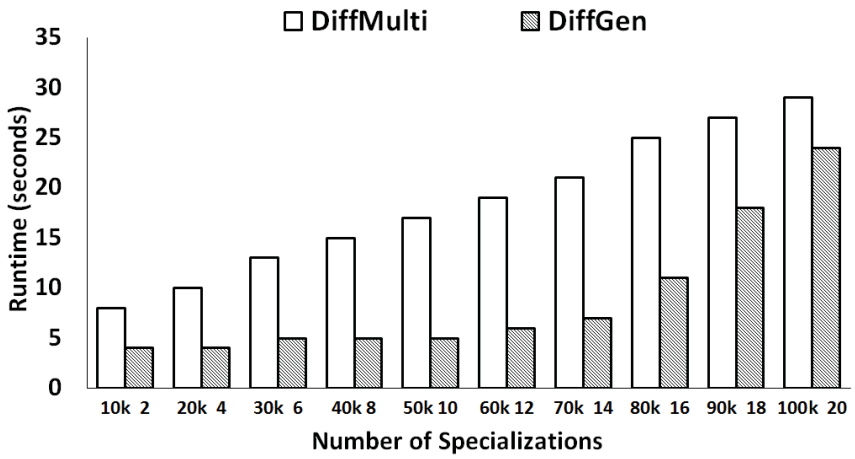


Figure 6.10: Comparing *DiffMulti* and *DiffGen* in terms of runtime

insensitive to the number of specializations as the increase in runtime is insignificant. For a clear visualization, we did not include *Mondrian* in Figure 6.10 because its runtime varied from a few minutes to nearly an hour for $20 \leq k \leq 100$. For the experiments in Figure 6.9, our method took 1 seconds to complete compared to a few minutes for *PrivBayes*, which was running on its default parameters. Particularly, the parameter that specifies the degree of Bayesian network k was set to 3. As k increases, the runtime of *PrivBayes* substantially increases; e.g., setting $k = 5$ causes *PrivBayes* to run for a few hours [141].

6.4.3 Scalability

The purpose of this experiment is to examine how our method scales to datasets with large number of records. We generated 5 variations of the raw *Adult* dataset, which has 45,222 records. For every record in *Adult*, we generate $\alpha - 1$ other records that contain some values drawn randomly from their pertinent attribute domain. By varying α we can generate a dataset with $\alpha \times 45,222$ records. We generated 5 datasets with number of records ranging from 200,000 to 1 million. We run *DiffMulti* once on each dataset, where the utility function is `Max` and $\epsilon = 1$. Figure 6.11 depicts

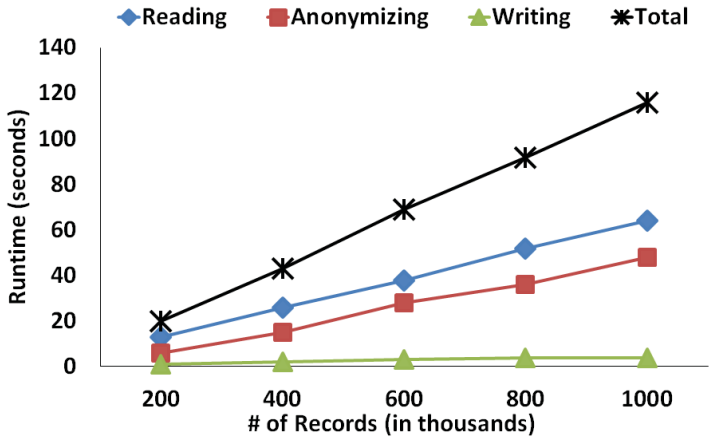


Figure 6.11: Scalability of *DiffMulti*

the runtime in seconds for reading the records, anonymizing the entire dataset, and writing the differentially-private multidimensionally-generalized records. Anonymizing an *Adult*-variant dataset with 1 million records finishes in less than 120 seconds. Figure 6.11 suggests that the runtime of *DiffMulti* increases reasonably as the number of records increases.

Overall. The privacy budget has a direct impact on the utility of the output data, and runtime is incremental with respect to the number of specializations and the number of input records.

6.5 Summary

In this research work, we propose a differentially-private and multidimensional generalization algorithm for publishing relational data. Publishing anonymous data in a *non-interactive* setting provides flexibility of usage to data recipients, as opposed to providing anonymous answers tailored to specific queries. We adopt differential privacy as our privacy model due to its independence of any attacker's background knowledge and insensitivity to the underlying data. Parallel to preserving privacy, the utility of the anonymous data is equally essential. Therefore, we effectively

generalize the raw data into multidimensional regions to minimize information loss. Experimental evaluation on a real-life dataset suggests that our proposed method is able to reduce information loss by at least one order of magnitude when compared with single-dimensional generalization, and improves data utility when compared with state-of-the-art private data release methods.

Chapter 7

Conclusion and Future Directions

The rapid development of information technology has rendered data exchange a compelling demand. Whether mandated by law or used for research purposes, data publishing has been a common practice in various sectors, such as hospitals, service providers, government agencies, and transportation authorities. However, the privacy of the individuals whose data is being published must not be compromised. The process of transforming person-specific data to an anonymous version is called *anonymization*. This thesis is motivated by real-life data-publishing incidences where individuals' identities and their sensitive information were not adequately protected in the released anonymous data.

Due to the emergence of new types of data mainly characterized by being *high-dimensional*, classical privacy protection models, such as k-anonymity, fail to accommodate such data types due to extreme data sparseness. Consequently, applying k-anonymity-based privacy models imposes high information loss, rendering the anonymous data unsuitable for fruitful analysis.

This thesis tackles the problem of anonymizing *high-dimensional* data for useful data mining, while simultaneously integrating different privacy concerns throughout the anonymization process. Particularly, we propose to anonymize *trajectory streams*, *static trajectories*, and *relational data* under rigorous privacy requirements.

Chapter 4 introduces us to the transient and time-varying data of trajectory streams. To tackle this challenge, we identify certain properties in trajectory streams and propose a dynamic anonymization solution based on incremental sliding windows. To our best knowledge, this is the first work to anonymize high-dimensional trajectory streams. In Chapter 5, we assume that trajectories of moving individuals have been collected and stored in advance. Hence, we study the problem of publishing trajectories under the strict *differential privacy* model. This is a challenging problem because trajectories are extremely sparse spatio-temporal data that requires special care throughout the anonymization process in order to result in high-quality anonymous data with reasonable computational cost. To overcome this obstacle, our proposed algorithm structures trajectories as a noisy prefix tree, which provides the desired compactness and data accessibility. Lastly, in Chapter 6, we observe the ever-expanding number of attributes in relational data and propose an efficient differentially-private solution that publishes anonymous relational data. We propose a workload-aware anonymization algorithm that can be tailored for high-quality data analysis, depending on the objective of the published data.

In conclusion, our extensive experimental evaluation suggests that, despite the inherent trade-off between data anonymity and utility, it is feasible to anonymize high-dimensional data, yet maintain high data utility for various analysis tasks. We acknowledge the profound effort by researchers in the privacy-preserving data publishing community, and this thesis is one step towards enhancing the quality of anonymous data.

The findings presented in this thesis have opened the door for profound research potential. A future data publishing problem could consider the practical case where the data spans multiple data holders, e.g., a full trajectory is composed of subtrajectories, each belongs to a different service provider, depending on the area of coverage. Releasing integrated data, rather than each part at a time, is greatly

beneficial for data analysts because they can see interesting patterns that may, otherwise, not have been discovered from analyzing only parts of the total integrated data. Having said that, the challenge remains in performing such anonymization without revealing detailed information about the data held by one data holder to another data holder who is not authorized to acquire such information, albeit about the same individual.

Another research direction could be to consider anonymizing data that consists of multiple types, e.g., transactions, sensory data, and texts. This is desirable because information about individuals is more likely to be collected from various sources. However, such data would be characterized by having a much larger and complex domain. Such complexity makes it challenging to impose a unified structure on the collected data and, consequently, requires non-trivial efforts in order to propose efficient algorithms.

Bibliography

[1] O. Abul, F. Bonchi, and M. Nanni. Never walk alone: Uncertainty for anonymity in moving objects databases. In *Proceedings of the 24th IEEE International Conference on Data Engineering (ICDE)*, pages 376–385, 2008.

[2] N. R. Adam and J. C. Worthmann. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys (CSUR)*, 21(4):515–556, 1989.

[3] C. C. Aggarwal. On k-anonymity and the curse of dimensionality. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, pages 901–909, 2005.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, pages 487–499, 1994.

[5] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE)*, pages 3–14, 1995.

[6] K. Al-Hussaeni, B. C. M. Fung, and W. K. Cheung. Privacy-preserving trajectory stream publishing. *Data & Knowledge Engineering (DKE)*, 94(A):89–109, 2014.

[7] K. Al-Hussaeni, B. C. M. Fung, G. Dagher, F. Iqbal, and E. G. Park. *SafePath*: Differentially-private publishing of passengers' trajectories. Under review.

[8] K. Al-Hussaeni, B. C. M. Fung, F. Iqbal, J. Liu, and P. C. K. Hung. Differentially-private multidimensional data publishing. Under 2nd revision.

[9] R. Assam, M. Hassani, and T. Seidl. Differential private trajectory protection of moving objects. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on GeoStreaming (IWGS)*, pages 68–77, 2012.

[10] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[11] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 1–16, 2002.

[12] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 633–634, 2002.

[13] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: A holistic solution to contingency table release. In *Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2007.

[14] M. Barbaro and T. Zeller. A face is exposed for AOL searcher no. 4417749. *New York Times*, August 9, 2006.

[15] R. J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 217–228, 2005.

[16] A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 609–618, 2008.

[17] F. Bonchi, L. V. Lakshmanan, and H. W. Wang. Trajectory anonymity in publishing personal mobility data. *ACM SIGKDD Explorations Newsletter*, 13(1):30–42, 2011.

[18] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *ACM SIGMOD Record*, 26(2):255–264, 1997.

[19] T. Brinkhoff. Generating traffic data. *IEEE Data Engineering Bulletin*, 26(2):19–25, 2003.

[20] M. Burger, M. van den Berg, A. Hegyi, B. D. Schutter, and J. Hellendoorn. Considerations for model-based traffic control. *Transportation Research Part C: Emerging Technologies*, 35:1–19, 2013.

[21] J.-W. Byun, Y. Sohn, E. Bertino, and N. Li. Secure anonymization for incremental datasets. In *Proceedings of the 3rd VLDB International Conference on Secure Data Management (SDM)*, pages 48–63, 2006.

[22] J. Cao, B. Carminati, E. Ferrari, and K. Lee Tan. Castle: A delay-constrained scheme for ks-anonymizing data streams. In *Proceedings of the 24th International Conference on Data Engineering (ICDE)*, pages 1376–1378, 2008.

[23] D. M. Carlisle, M. L. Rodrian, and C. L. Diamond. California inpatient data reporting manual, medical information reporting for california (5th ed). Technical report, Office of Statewide Health Planning and Development, July 2007.

[24] T.-H. H. Chan, E. Shi, and D. Song. *Privacy-Preserving Stream Aggregation with Fault Tolerance*, pages 200–214. Springer Berlin Heidelberg, 2012.

[25] S. Chawla, C. Dwork, F. McSherry, A. Smith, and H. Wee. Toward privacy in public databases. In *Proceedings of the 2nd International Conference on Theory of Cryptography (TCC)*, pages 363–385, 2005.

[26] R. Chen, B. C. M. Fung, B. C. Desai, and N. M. Sossou. Differentially private transit data publication: A case study on the montreal transportation system. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 213–221, 2012.

[27] R. Chen, B. C. M. Fung, N. Mohammed, and B. C. Desai. Privacy-preserving trajectory data publishing by local suppression. *Information Sciences: Special Issue on Data Mining for Information Security*, 231:83–97, 2013.

[28] R. Chen, B. C. M. Fung, P. S. Yu, and B. C. Desai. Correlated network data publication via differential privacy. *The VLDB Journal*, 23(4):653–676, 2014.

[29] R. Chen, N. Mohammed, B. C. M. Fung, B. C. Desai, and L. Xiong. Publishing set-valued data via differential privacy. *Proceedings of the VLDB Endowment*, 4(11):1087–1098, 2011.

[30] A. E. Cicek, M. E. Nergiz, and Y. Saygin. Ensuring location diversity in privacy-preserving spatio-temporal data publishing. *The VLDB Journal*, 23(4):609–625, 2014.

[31] G. Cormode, C. Procopiuc, D. Srivastava, and T. T. L. Tran. Differentially private summaries for sparse data. In *Proceedings of the 15th International Conference on Database Theory (ICDT)*, pages 299–311, 2012.

[32] G. Cormode, D. Srivastava, N. Li, and T. Li. Minimizing minimality and maximizing utility: Analyzing method-based attacks on anonymized data. *Proceedings of the VLDB Endowment*, 3(1-2):1045–1056, 2010.

[33] B. Ding, M. Winslett, J. Han, and Z. Li. Differentially private data cubes: Optimizing noise sources and consistency. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 217–228, 2011.

[34] C. Dwork. Differential privacy. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II (ICALP)*, pages 1–12, 2006.

[35] C. Dwork. Differential privacy: A survey of results. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation (TAMC)*, pages 1–19, 2008.

[36] C. Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[37] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the 3rd Conference on Theory of Cryptography (TCC)*, pages 265–284, 2006.

[38] C. Dwork, M. Naor, T. Pitassi, G. N. Rothblum, and S. Yekhanin. Pan-private streaming algorithms. In *Proceedings of the 1st Symposium on Innovations in Computer Science (ICS)*, 2010.

[39] L. Fan, L. Xiong, and V. Sunderam. Differentially private multi-dimensional time series release for traffic monitoring. In *Proceedings of the 27th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy XXVII - Volume 7964 (DBSec)*, pages 33–48, 2013.

[40] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[41] A. Friedman and A. Schuster. Data mining with differential privacy. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 493–502, 2010.

[42] B. C. M. Fung, K. Al-Hussaeni, and M. Cao. Preserving RFID data privacy. In *Proceedings of the IEEE International Conference on RFID*, pages 200–207, 2009.

[43] B. C. M. Fung, T. Trojer, P. C. K. Hung, L. Xiong, K. Al-Hussaeni, and R. Dssouli. Service-oriented architecture for high-dimensional private data mashup. *IEEE Transactions on Services Computing*, 5(3):373–386, 2012.

[44] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (CSUR)*, 42(4):14:1–14:53, 2010.

[45] B. C. M. Fung, K. Wang, and P. S. Yu. Anonymizing classification data for privacy preservation. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(5):711–725, 2007.

[46] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: A review. *ACM SIGMOD Record*, 34:18–26, 2005.

[47] S. Gambs, M.-O. Killijian, and M. N. n. del Prado Cortez. Show me how you move and i will tell you who you are. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*, SPRINGL '10, pages 34–41, 2010.

[48] S. R. Ganta, S. P. Kasiviswanathan, and A. Smith. Composition attacks and auxiliary information in data privacy. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 265–273, 2008.

[49] M. Ghasemzadeh, B. C. M. Fung, R. Chen, and A. Awasthi. Anonymizing trajectory data for passenger flow analysis. *Transportation Research Part C: Emerging Technologies*, 39:63–79, 2014.

[50] G. Ghinita, Y. Tao, and P. Kalnis. On the anonymization of sparse high-dimensional data. In *Proceedings of the 24th IEEE International Conference on Data Engineering (ICDE)*, pages 715–724, 2008.

[51] G. Gidofalvi, X. Huang, and T. B. Pedersen. Privacy-preserving data mining on moving object trajectories. In *Proceedings of the International Conference on Mobile Data Management (MDM)*, pages 60–68, 2007.

[52] A. Gkoulalas-Divanis and G. Loukides. Utility-guided clustering-based transaction data anonymization. *Transactions on Data Privacy*, 5(1):223–251, 2012.

[53] L. Golab and M. T. Özsu. Issues in data stream management. *ACM SIGMOD Record*, 32(2):5–14, 2003.

[54] P. Golle. Revisiting the uniqueness of simple demographics in the us population. In *Proceedings of the 5th ACM Workshop on Privacy in Electronic Society (WPES)*, pages 77–80, 2006.

[55] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and analyzing massive RFID data sets. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, pages 83–92, 2006.

[56] P. J. Haas and A. N. Swami. Sequential sampling procedures for query size estimation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 341–350, 1992.

[57] K. Hafner. And if You Liked the Movie, a Netflix Contest May Reward You Handsomely. *New York Times*, October 6, 2006.

[58] K. Hafner and T. Zeller. Researchers yearn to use AOL logs, but they hesitate. *New York Times*, August 23, 2006.

[59] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *ACM SIGMOD Record*, 29(2):1–12, 2000.

[60] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proceedings of the VLDB Endowment*, 3(1-2):1021–1032, 2010.

[61] X. He, G. Cormode, A. Machanavajjhala, C. M. Procopiuc, and D. Srivastava. Dpt: Differentially private trajectory synthesis using hierarchical reference systems. *Proceedings of the VLDB Endowment*, 8(11):1154–1165, 2015.

[62] Y. He, S. Barman, D. Wang, and J. F. Naughton. On the complexity of privacy-preserving complex event processing. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, pages 165–174, 2011.

[63] Y. He and J. F. Naughton. Anonymization of set-valued data via top-down, local generalization. *Proceedings of the VLDB Endowment*, 2(1):934–945, 2009.

[64] S.-S. Ho. Preserving privacy for moving objects data mining. In *Proceedings of the IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 135–137, 2012.

[65] S.-S. Ho and S. Ruan. Preserving privacy for interesting location pattern mining from trajectory data. *Transactions on Data Privacy*, 6(1):87–106, 2013.

[66] Y. Hong, J. Vaidya, H. Lu, and M. Wu. Differentially private search log sanitization with optimal output utility. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT)*, pages 50–61, 2012.

[67] H. Hu, J. Xu, S. T. On, J. Du, and J. K.-Y. Ng. Privacy-aware location data publishing. *ACM Transactions on Database Systems (TODS)*, 35:18:1–18:42, 2010.

[68] V. S. Iyengar. Transforming data to satisfy privacy constraints. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 279–288, 2002.

[69] K. Jiang, D. Shao, S. Bressan, T. Kister, and K.-L. Tan. Publishing trajectories with differential privacy guarantees. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 12:1–12:12, 2013.

[70] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

[71] A. Juels. RFID security and privacy: a research survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, 2006.

[72] D. Kifer. Attacks on privacy and definetti's theorem. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 127–138, 2009.

[73] D. Kifer and B.-R. Lin. Towards an axiomatization of statistical privacy and utility. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 147–158, 2010.

[74] J. Lee and C. W. Clifton. Top-k frequent itemsets via differentially private fp-trees. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 931–940, 2014.

[75] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 49–60, 2005.

[76] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, pages 25–35, 2006.

[77] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Workload-aware anonymization. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 277–286, 2006.

[78] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Workload-aware anonymization techniques for large-scale datasets. *ACM Transactions on Database Systems (TODS)*, 33(3):17:1–17:47, 2008.

[79] D. Leoni. Non-interactive differential privacy: A survey. In *Proceedings of the 1st International Workshop on Open Data (WOD)*, pages 40–52, 2012.

[80] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 123–134, 2010.

[81] F. Li, J. Sun, S. Papadimitriou, G. Mihaila, and I. Stanoi. Hiding in the crowd: Privacy preservation on evolving streams through correlation tracking. In *Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE)*, pages 686–695, 2007.

[82] J. Li, A. W.-C. Fu, H. He, J. Chen, H. Jin, D. McAullay, G. Williams, R. Sparks, and C. Kelman. Mining risk patterns in medical data. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pages 770–775, 2005.

[83] J. Li, B. C. Ooi, and W. Wang. Anonymizing streaming data for privacy protection. In *Proceedings of the 24th IEEE International Conference on Data Engineering (ICDE)*, pages 1367–1369, 2008.

[84] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE)*, pages 106–115, 2007.

[85] N. Li, W. Qardaji, D. Su, and J. Cao. Privbasis: Frequent itemset mining with differential privacy. *Proceedings of the VLDB Endowment*, 5(11):1340–1351, 2012.

[86] X. Li, J. Han, J.-G. Lee, and H. Gonzalez. Traffic density-based discovery of hot routes in road networks. In *Proceedings of the 10th International Conference on Advances in Spatial and Temporal Databases (SSTD)*, pages 441–459, 2007.

[87] C. Liu, S. Chakraborty, and P. Mittal. Dependence makes you vulnerable: Differential privacy under dependent tuples. In *Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS)*, 2016.

[88] D. Luper, D. Cameron, J. Miller, and H. R. Arabnia. Spatial and temporal target association through semantic analysis and gps data mining. In *Proceedings of the 5th International Conference on Information and Knowledge Engineering (IKE)*, volume 7, pages 251–257, 2007.

[89] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. In *Proceedings of the 22nd IEEE International Conference on Data Engineering (ICDE)*, pages 24–35, 2006.

[90] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1), 2007.

[91] J. M. Mateo-Sanz, A. Martínez-Ballesté, and J. Domingo-Ferrer. Fast generation of accurate synthetic microdata. In *Proceedings of Privacy in Statistical Databases: CASC Project International Workshop (PSD)*, volume 3050 of *Lecture Notes in Computer Science*, pages 298–306. Springer Berlin Heidelberg, 2004.

[92] T. McGhee. Gps technology tracks employees. *The Denver Post*, December 8, 2006.

[93] F. McSherry. Privacy integrated queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 19–30, 2009.

[94] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 94–103, 2007.

[95] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 223–228, 2004.

[96] N. Mohammed, R. Chen, B. C. M. Fung, and P. S. Yu. Differentially private data release for data mining. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 493–501, 2011.

[97] N. Mohammed, B. C. M. Fung, and M. Debbabi. Walking in the crowd: Anonymizing trajectory data for pattern analysis. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)*, pages 1441–1444, 2009.

[98] N. Mohammed, B. C. M. Fung, P. C. K. Hung, and C.-k. Lee. Anonymizing healthcare data: A case study on the blood transfusion service. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1285–1294, 2009.

[99] A. Monreale, G. Andrienko, N. Andrienko, F. Giannotti, D. Pedreschi, S. Rinzivillo, and S. Wrobel. Movement data anonymity through generalization. *Transactions on Data Privacy*, 3(2):91–121, 2010.

[100] A. Narayan, A. Feldman, A. Papadimitriou, and A. Haeberlen. Verifiable differential privacy. In *Proceedings of the 10th European Conference on Computer Systems (EuroSys)*, pages 28:1–28:14, 2015.

[101] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 111–125, 2008.

[102] C. Negroni. Tracking your wi-fi trail. *New York Times*, March 2011.

[103] M. E. Nergiz, M. Atzori, Y. Saygın, and B. Güç. Towards trajectory anonymization: A generalization-based approach. *Transactions on Data Privacy*, 2(1):47–75, 2009.

[104] R. G. Pensa, A. Monreale, F. Pinelli, and D. Pedreschi. Pattern-preserving k-anonymization of sequences and its application to mobility data mining. In *Proceedings of the 1st International Workshop on Privacy in Location-Based Applications*, 2008.

[105] V. Primault, S. B. Mokhtar, C. Lauradoux, and L. Brunie. Time distortion anonymization for the publication of mobility data with high utility. In *Proceedings of the 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 539–546, 2015.

[106] W. Qardaji and N. Li. Recursive partitioning and summarization: A practical framework for differentially private data publishing. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 38–39, 2012.

[107] W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *Proceedings of the VLDB Endowment*, 6(14):1954–1965, 2013.

[108] W. Qardaji, W. Yang, and N. Li. Priview: Practical differentially private release of marginal contingency tables. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1435–1446, 2014.

[109] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[110] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(6):1010–1027, 2001.

[111] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, 1998.

[112] R. Sherkat, J. Li, and N. Mamoulis. Efficient time-stamped event sequence anonymization. *ACM Transactions on the Web (TWEB)*, 8(1):4:1–4:53, 2013.

[113] L. Sweeney. Datafly: A system for providing anonymity in medical data. In *Proceedings of the IFIP TC11 WG11.3 11th International Conference on Database Securty XI: Status and Prospects*, pages 356–381, 1998.

[114] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):571–588, 2002.

[115] L. Sweeney. K-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[116] L.-A. Tang, Y. Zheng, J. Yuan, J. Han, A. Leung, W.-C. Peng, and T. L. Porta. A framework of traveling companion discovery on trajectory data streams. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(1):3:1–3:34, 2014.

[117] M. Terrovitis and N. Mamoulis. Privacy preservation in the publication of trajectories. In *Proceedings of the 9th International Conference on Mobile Data Management (MDM)*, pages 65–72, 2008.

[118] M. Terrovitis, N. Mamoulis, and P. Kalnis. Privacy-preserving anonymization of set-valued data. *Proceedings of the VLDB Endowment*, 1(1):115–125, 2008.

[119] M. Terrovitis, N. Mamoulis, and P. Kalnis. Local and global recoding methods for anonymizing set-valued data. *The VLDB Journal*, 20(1):83–106, 2011.

[120] J. Wang, S. Liu, and Y. Li. A review of differential privacy in individual data release. *International Journal of Distributed Sensor Networks*, 2015:1:1–1:1, 2016.

[121] K. Wang, B. C. M. Fung, and P. S. Yu. Handicapping attacker's confidence: An alternative to k-anonymization. *Knowledge and Information Systems (KAIS)*, 11(3):345–368, 2007.

[122] K. Wang, Y. Xu, R. C.-W. Wong, and A. W.-C. Fu. Anonymizing temporal data. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM)*, pages 1109 –1114, 2010.

[123] P. Wang, L. Zhao, J. Lu, and J. Yang. Sanatomy: Privacy preserving publishing of data streams via anatomy. In *Proceedings of the 3rd International Symposium on Information Processing (ISIP)*, pages 54–57, 2010.

[124] S.-W. Wang, W.-H. Chen, C.-S. Ong, L. Liu, and Y.-W. Chuang. RFID application in hospitals: A case study on a demonstration RFID project in a taiwan hospital. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS)*, volume 8, pages 184a–184a, 2006.

[125] S. M. Weiss and C. A. Kulikowski. *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1991.

[126] R. C.-W. Wong, A. W.-C. Fu, K. Wang, and J. Pei. Minimality attack in privacy preserving data publishing. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, pages 543–554, 2007.

[127] R. C.-W. Wong, J. Li, A. W.-C. Fu, and K. Wang. (α, k)-anonymity: An enhanced k-anonymity model for privacy preserving data publishing. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 754–759, 2006.

[128] X. Xiao, G. Bender, M. Hay, and J. Gehrke. ireduct: Differential privacy with reduced relative errors. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 229–240, 2011.

[129] X. Xiao and Y. Tao. M-invariance: Towards privacy preserving re-publication of dynamic datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 689–700, 2007.

[130] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 23(8):1200–1214, 2011.

[131] X. Xiao, K. Yi, and Y. Tao. The hardness and approximation algorithms for l-diversity. In *Proceedings of the 13th International Conference on Extending Database Technology (EDBT)*, pages 135–146, 2010.

[132] Y. Xiao and L. Xiong. Protecting locations with differential privacy under temporal correlations. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1298–1309, 2015.

[133] Y. Xiao, L. Xiong, L. Fan, S. Goryczka, and H. Li. Dpcube: Differentially private histogram release through multidimensional partitioning. *Transactions on Data Privacy*, 7(3):195–222, 2014.

[134] Y. Xiao, L. Xiong, and C. Yuan. Differentially private data release through multidimensional partitioning. In *Proceedings of the 7th VLDB Conference on Secure Data Management (SDM)*, pages 150–168, 2010.

[135] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A. W.-C. Fu. Utility-based anonymization using local recoding. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 785–790, 2006.

[136] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett. Differentially private histogram publication. *The VLDB Journal*, 22(6):797–822, 2013.

[137] Y. Xu, B. C. M. Fung, K. Wang, A. W. C. Fu, and J. Pei. Publishing sensitive transactions for itemset utility. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM)*, 2008.

[138] Y. Xu, K. Wang, A. W.-C. Fu, and P. S. Yu. Anonymizing transaction databases for publication. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 767–775, 2008.

[139] R. Yarovoy, F. Bonchi, L. V. S. Lakshmanan, and W. H. Wang. Anonymizing moving objects: how to hide a mob in a crowd? In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT)*, pages 72–83, 2009.

[140] C. Zeng, J. F. Naughton, and J.-Y. Cai. On differentially private frequent itemset mining. *Proceedings of the VLDB Endowment*, 6(1):25–36, 2012.

[141] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: Private data release via bayesian networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1423–1434, 2014.

[142] J. Zhang, J. Yang, J. Zhang, and Y. Yuan. Kids:k-anonymization data stream base on sliding window. In *Proceedings of the 2nd International Conference*

on Future Computer and Communication (ICFCC), volume 2, pages V2–311–V2–316, 2010.

[143] Y. Zheng, N. J. Yuan, K. Zheng, and S. Shang. On discovery of gathering patterns from trajectories. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 242–253, 2013.

[144] B. Zhou, Y. Han, J. Pei, B. Jiang, Y. Tao, and Y. Jia. Continuous privacy preserving publishing of data streams. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT)*, pages 648–659, 2009.