Implementing Differential Privacy for Privacy Preserving Trajectory Data Publication in Large-Scale Wireless Networks

Caleb Stroud

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

29 June 2018
Blacksburg, VA

Keywords: ε-differential privacy, differential privacy, privacy preserving trajectory data publication, PPTDP, privacy preserving data publication, PPDP

Implementing Differential Privacy for Privacy Preserving Trajectory Data Publication in Large-Scale Wireless Networks

Caleb Stroud

ABSTRACT

Wireless networks collect vast amounts of log data concerning usage of the network. This data aids in informing operational needs related to performance, maintenance, etc., but it is also useful for outside researchers in analyzing network operation and user trends. Releasing such information to these outside researchers poses a threat to privacy of users. The dueling need for utility and privacy must be addressed. This thesis studies the concept of differential privacy for fulfillment of these goals of releasing high utility data to researchers while maintaining user privacy. The focus is specifically on physical user trajectories in authentication manager log data since this is a rich type of data that is useful for trend analysis. Authentication manager log data is produced when devices connect to physical access points (APs) and trajectories are sequences of these spatiotemporal connections from one AP to another for the same device. The fulfillment of this goal is pursued with a variable length n-gram model that creates a synthetic database which can be easily ingested by researchers. We found that there are shortcomings to the algorithm chosen in specific application to the data chosen, but differential privacy itself can still be used to release sanitized datasets while maintaining utility if the data has a low sparsity.

Implementing Differential Privacy for Privacy Preserving Trajectory Data Publication in Large-Scale Wireless Networks

Caleb Stroud

GENERAL AUDIENCE ABSTRACT

Wireless internet networks store historical logs of user device interaction with it. For example, when a phone or other wireless device connects, data is stored by the Internet Service Provider (ISP) about the device, username, time, and location of connection. A database of this type of data can help researchers analyze user trends in the network, but the data contains personally identifiable information for the users. We propose and analyze an algorithm which can release this data in a high utility manner for the researchers, yet maintain user privacy. This is based on a verifiable approach to privacy called differential privacy. This algorithm is found to provide utility and privacy protection for datasets with many users compared to the size of the network.

ACKNOWLEDGEMENTS

I would first like to thank my advisor and committee members, Dr. Joe Tront, Dr. Dave Raymond, and Dr. Patrick Schaumont for guiding me through this process. This would not have been possible if it had not been for the Virginia Tech ITSL (IT Security Lab) led by Randy Marchany, and all the help I received from others there. This includes Mark DeYoung, Mike Cantrell, and Brad Tilley. To all my friends and family who have stuck with me through this process, I thank you for your patience, understanding, and willingness to provide insight.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF EQUATIONS

1 Introduction

The high usage of technology in today's culture provides the opportunity for the collection of vast amounts of increasingly detailed user data. Specifically, within a wireless network, there are many logged events including those recorded in authentication logs (authentication manager (authmgr), Radius Remote Authentication Dial-In User Service [RADIUS], etc.), Internet Protocol flow metadata, system logs, performance telemetry, intrusion alerts, and others. This large number of data sources has brought about many studies concerning network data. Previous work has included characterizing user behavior and network performance in a public-area wireless network [1]. Others have analyzed traffic usage on a university campus [2]. There has also been a model created for predictive mobility management [3]. Another research topic has been device characterization, such as studying the difference between laptop and smartphone traces [4]. The focus of this paper is on data from large networks. This area has included studies on eduroam, namely characterizing educational wireless traffic in large-scale wireless roaming networks (eduroam) [5]. Such research has also included examining various network flows to study network usage in a network with 550 access points [6], analyzing user sessions from network controller syslog events on the University of Massachusetts with 4,500 access points and 35,000 students [7], and using IP traffic to cluster traffic in a large university [8]. Most of these studies took some precautions to sanitize the data, but many of these techniques were very primitive. Nonetheless, they show the wide range and high demand for wireless network data to be used for usage analysis of networks.

1.1 Privacy vs Utility

Releasing this type of data brings about two competing goals: maintaining the privacy of users in the dataset and maintaining utility of the data. Often these two goals originate from competing groups, so the data holder must choose how to fulfill the desires of various groups of users.

Privacy is a seemingly elusive concept that has had much coverage recently. It is both desired by most individuals, since they do not want every aspect of their personal lives broadcasted to the public, and seemingly ignored in other cases, when the benefits of an action outweigh the cost to privacy. This shows that privacy both means different things to different people and could change for each individual based on the circumstance [9]. Thus creating privacy mechanisms which adhere to an ever evolving concept of and desire for privacy is difficult.

Privacy is also a concern of policy makers and lawmakers. The National Institute of Standards and Technology (NIST) created a recommendation document advising government personnel on the informed usage of de-identification techniques [10]. Specific to university campuses, there is often an Institutional Review Board (IRB) that dictates how data can be used. At Virginia Tech, the IRB requires that certain identifiable information must not be released at all and other information must not be released together. This can be easily achieved with simple de-identification techniques, but this section and Chapter 2 will show that more is required to actually protect privacy. The existence of these laws and regulations, though, indicate the challenge related to privacy and the existence of ethical considerations when releasing personal information.

Utility, or data utility, is closeness of certain statistics that can be derived from the data. This is different than data quality, which is the closeness of abstract qualities to the original dataset [10]. This distinction is important because most data anonymization techniques perturb the original dataset in some way, which may lead to poor quality, but high utility. Utility is the key factor looked at since that is what provides useful statistical information about the dataset, specifically keeping the utility of the anonymized dataset as close as possible to the original raw dataset. Since most anonymization schemes involve some sort of perturbation of the original data, this may reduce data quality but does not necessarily have to decrease data utility.

There is often a contention between these concepts of privacy and utility. Data cannot be fully anonymized as there would be absolutely no utility left. Conversely, if the data has full utility, there would be no privacy. Thus there must be a balance between the two. This balance, though, does not have to be a linear relationship. A relatively private algorithm can give high utility for certain types of expected queries on the data.

1.2 Motivation

Often cited in privacy research are a few famous privacy breaches to show that proper data anonymization is hard [9]. In the GIC re-identification attack, the Group Insurance Commission in Massachusetts published an “anonymized” dataset of medical encounter data because of a health regulation in that state. Latanya Sweeney bought a Voter

Registration List and was able to cross link the medical information for many individuals, including the former governor of Massachusetts, William Weld [11]. In the AOL debacle, the company AOL decided to publish anonymized query history of individuals. The New York Times was able to identify one of the users, No. 4417749, as Thelma Arnold simply by analyzing all the queries made by this “anonymized” individual [12]. In the Netflix prize case, Netflix released a dataset of 500,000 anonymized data records with all personal information removed in order to create a better recommendation system. Then, researchers Narayanan and Shmatikov correlated this dataset with the public IMDb rating database, where individuals’ names and ratings can be found, since users rated movies about the same time on both services [13]. They found that 99% of Netflix subscribers in the released dataset could be identified with just 8 public IMDb movie ratings. The data in all these attacks were anonymized correctly based off of the anonymization scheme chosen. Thus the schemes themselves were not adequate to prevent against all attacks, specifically the ones described.

The data in the research just described were all tabular data. According to de Montjoye et al., 95% of individuals in a dataset can be identified with only four spatiotemporal points and this uniqueness holds even with coarse datasets [14]. It is important to note this is based on the ability to use outside, or background, information, addressed in Sections 2.2 and 2.3, but this still shows that along with vulnerabilities available in all data, more complex data presents even more avenues for attack and thus data release. Such data includes trajectory data, which is a sequence of spatiotemporal points. This thesis focuses on trajectory data in wireless networks. It has been shown that basic sanitization algorithms used on network user association logs are not sufficient [15]. The attackers built user signatures from the anonymized data from a network of 1300 Aruba APs and were able to identify user identity with probability 70% based on association with background information.

With the tension between privacy and utility in mind, there needs to be a meaningful, strong definition of privacy that also is practical and computationally feasible. The next section describes the specific purpose and application sought for this research endeavor.

1.3 Purpose

The Virginia Tech Division of Information Technology maintains networking and computational resources for affiliated personnel. Among many other tasks, this includes acting as an Internet Service Provider (ISP). All of these resources collect information related to their usage. These logs are collected in a large scale data aggregation system. Such information could be extremely useful to researchers, and there is an effort to release logs to researchers in a private and utility preserving manner. Trajectory data is rich with information for user movement, density analysis, prediction algorithms, and much more. With the publishing of sanitized versions of multiple log sources comes many challenges. The main issue related to privacy is correlation between datasets. This comes in two forms. First, in the process of anonymizing each dataset, correlated events must remain. For example, when a user first connects to the wireless network, there is a certain number and order of logs created to authenticate and connect the user. If these logs do not exist in the sanitized datasets, this destroys utility. Second, each dataset must be anonymized in such a way that re-identification or correlation attacks are not possible between datasets.

In order to venture towards publishing sanitized versions of the many log sources, this thesis takes a gradual approach and focuses on anonymizing data from one such data source, specifically authentication manager (authmgr) logs. These occur every time a device, whether a laptop computer, cell phone, or other portable device, connects to a wireless access point (AP) within the Virginia Tech wireless network. Figure 1.1 shows the basic structure of one authentication manager log. Note that there is a location (access point name), time (timestamp), and identifiers (MAC address and Username). Since this data contains the locations of users, it is natural that such data would be used for such queries as movement trends or location counts for user density counts. Thus, this research takes the approach of anonymizing trajectories of identifiers (MAC/username combinations) within the dataset. This means the data must be converted from tabular data into a trajectory dataset. This step is described in Section 5.2.

Timestamp	**Access Point Name**	**AP Room**	**Building**	**Source MAC Addr**	**Username**
2017-06-05T04:20:33.025Z	TOR-1040-AP01B	1040	Torgerson Hall	12:34:56:78:9a:bc	coolpid@vt.edu

Figure 1.1 Authentication Manager log format

Note the difference between a sequence and a trajectory. A sequence is a time-ordered list of locations. A trajectory is a list of spatiotemporal points, or time-ordered events. The time portion adds important dimensional information to the data. Instead of simply knowing the locations of users, the time at the location can also be known. This aids in trend analysis over time. Note that an algorithm that works with trajectory data can work for sequence data, just set the time portion to zero, but not the inverse.

There are a few inherent data limitations for authmgr data in the network chosen. The exact geographical location of the items in the universe of locations are also not known. If data such as latitude and longitude were used, the whereabouts of each location would be known and thus relational qualities between them, such as relative distance and absolute position, could be calculated. Thus the algorithm cannot use spatial relational data to improve performance. Also, location is based off of which AP the user is connected to, which means the device is within the radius of the AP signal. This does not give an exact position and is sometimes difficult to discover the exact position of the device [16]. Associated with this is elevation information. Just because a user is connected to an access point on the third floor of a building does not always mean he/she is on that floor. Despite this, the AP connection information does give approximate locations which on a large university campus is good enough for such tasks as movement analysis.

Once trajectories are created, a differentially private algorithm uses variable length n-grams to construct a synthetic database. The purpose of this study is to analyze the usage of differential privacy to the specific case of releasing wireless logs to the research community. The next section describes specific contributions of this thesis.

1.4 Contributions

The contributions of this research endeavor are as follows:

- Describe the implementation details of a differentially private algorithm with a specific application of privacy preserving release of the trajectories derived from wireless network data, though the implementation could be used for other similarly structured data.
- Instead of taking as input and outputting a sequence dataset, this implementation reads in and outputs tabular data containing sequential information. Thus a simple method for trajectory creation and disassembly is introduced.
- The data focus is spatiotemporal points with explicit timestamps and locations. Most previous work focuses on time-ordered sequences.
- Creation of a framework for testing differentially private trajectory algorithms.
- Analysis of differential privacy for its use in high dimensional data. This concerns ease of implementation and utility results, specifically with counting queries and frequent sequential pattern mining.

The rest of the thesis is organized as follows. Chapter 2 describes related work concerning de-identification and anonymization over time, differential privacy, and the specific application of privacy preserving trajectory data publication. Chapter 3 describes the necessary differential privacy and n-gram model background information related to the algorithmic design. Chapter 4 gives an overview of the algorithm designed and the approach used in this thesis, while Chapter 5 contains the implementation of this algorithm. Chapter 6 describes results taken from running the implementation, namely an analysis of parameter choice, utility results, and algorithmic design. Chapter 7 ends with conclusions and future work. Appendix A gives guidelines for installation and usage of the software written.

2 Related Work

This section will begin with some definitions of terms used. The term de-identification will be used as defined in the NIST document as methods and algorithms used to remove, modify, or obfuscate data in order to protect individuals from information risks [10]. This is in comparison with the term anonymization which implies there can be no linkage to an actual identity. Some authors have said true anonymization cannot be attained, but this thesis will use the term to describe algorithms with a formal sense of privacy versus those that just de-identify data. This thesis will use the word sanitized to refer to both de-identified and anonymized datasets. De-identification and anonymization algorithms and methods aim to preserve both privacy and utility, as described in the introduction.

A raw dataset (or database) is a set of data taken directly from individuals and is the input to sanitization algorithms. The output can either be the answer to a query, or a synthetic database, which is a dataset that generally has the same format and information as the raw dataset but which has been sanitized.

This chapter describes related work. The first three sections detail related work in sanitization schemes (Sections 2.1, 2.2, and 2.3), while the fourth describes sanitization specifically related to database release (Section 2.4) and the last section anonymization algorithms (Section 2.5), before concluding (Section 2.6).

2.1 Statistical Disclosure Control

A preliminary method to release data was to release statistics about the data. A form of data release that has been widely used for at least a few decades is Statistical Disclosure Control (SDC) or privacy-preserving data analysis [17], a survey of which can be found by the National Research Council [18]. Releasing statistics is useful for some contexts, but neither guarantees privacy nor utility in all situations. SDC is not inherently inadequate, it is just that a majority the algorithms are susceptible to various attacks. Thus some formalization is needed.

2.2 Syntactic Anonymization

With the increased release of whole datasets, preliminary SDC schemes were not enough and new methods had to be introduced, namely syntactic or partition-based schemes [19]. Very basic forms of data anonymization have been used over the years, but these basic techniques only de-identify data. Specifically, these techniques consists of simply removing or hashing directly identifiable information such as names or addresses. At first they only dealt with direct identifiers, but it was later seen that private data could still be ascertained from indirect identifiers, or quasi identifiers [11]. These techniques may protect against the very basic attacks, but as discussed by Sweeney, they do not provide any guarantee of privacy against most attacks such as re-identification by linking. For example, as described in Chapter 1 Sweeney was able to identify the medical records of the governor of Massachusetts by linking voter registration records to an “anonymized” hospital visit dataset from the Group Insurance Commission [11].

K-anonymity was introduced by Sweeney in the early 2000s to remedy some of the shortfalls of simple suppression techniques [11], [20]. It was specifically produced to protect against linkage attacks used to reveal the identity or other personal properties of an individual given some data about them from a database. K-anonymity generally works on tabular data where there are rows which contain both direct identifiers and quasi identifiers. The concept is that an individual cannot be distinguished from k-1 other individuals within the dataset. This group of k records is called an equivalence class [21]. It works by employing suppression and generalization to mutate the data into a de-identified form. Suppression is the removal of attributes whereas generalization is the process of making attributes less unique to achieve a certain level of k-anonymity. Generally direct identifiers are suppressed and quasi identifiers are suppressed or generalized.

Other methods build on k-anonymity, including the following. ℓ-diversity protects against attribute disclosure by improving attribute diversity by requiring that each equivalence class has ℓ values for each sensitive attribute [22]. t-closeness takes a slightly different approach than ℓ-diversity and requires statistical closeness to the original dataset in that the distribution of any sensitive attribute be no more that t different than the distribution in the original dataset [21]. A newer method, (k, ε)-anonymity, ensures there are semantic similarity between sensitive attributes, requiring k-anonymity and ε-similar, which measure the closeness of sensitive values with regards to hierarchical trees [23]. That paper along with Wei, et al.'s paper [24] give a general overview of some other methods as well. These schemes are labeled as partition-based because they rely on hiding data in buckets to achieve privacy. They can also be described as syntactic since privacy relies on the structure of the resulting sanitized dataset.

Partition-based schemes do not provide protection if equivalence classes do not have diversity or the attacker has sufficient background knowledge. Background knowledge is any data other than the de-identified dataset that is used in some manner to reveal attributes or identities of persons in the de-identified dataset. This makes it susceptible to many attacks. The composition attack combines data from independently released sanitized datasets to breach privacy [25]. The deFinetti attack requires only a small amount of background knowledge [26]. It uses Bayesian networks, which are probabilistic graphical models, to correlate non-sensitive attributes to sensitive attributes then uses the deFinetti Representation Theorem, which describes the independence of distributions, to predict the sensitive attribute of someone in the dataset. Thus, seemingly non-sensitive attributes are used to breach privacy. Instead of using background knowledge, the foreground knowledge attack uses the inherent requirement of utility of data to find patterns in the resulting dataset to breach privacy [27]. Specifically, it deduces patterns from the resulting bucketed groups making it easier to identify attributes of persons in the dataset. ℓ-diversity, t-closeness, and others have been designed to not be susceptible to some of these attacks, but because of the deterministic structure of the resulting datasets and reliance on lack of background knowledge from adversaries, it is presumed that all such partition-based schemes are susceptible to various such attacks.

These approaches provide some arbitrary level of protection. Though they could be sufficient in some cases, the main problem is that their privacy guarantees are based on the amount of background knowledge of the attacker. This means that for any additional privacy mechanism or technique, there could always be an attacker that has enough background information to breach privacy. There is no formal definition of the privacy guarantee. This is where differential privacy comes in.

2.3 Differential Privacy

A few years after Sweeney published her paper, Dwork at Microsoft Research introduced the concept of differential privacy to provide a stronger notion of privacy [28]. This definition came from work in the fields of SDC, as well as theoretical computer science. Whereas k-anonymity is built on a mechanism that de-identifies data, differential privacy provides a definition of privacy and mechanisms can be created to guarantee this definition. Thus the term differential privacy is used to refer both to this definition, formalized in Section 3.1, and methods of achieving this definition. In this thesis, the term differential privacy is used for the former, and differentially private algorithm for the latter.

First, the setting must be described for differential privacy. Assume that there are three individuals. One is a trusted entity, the data curator, who holds the raw data and applies the differentially private algorithm. The second is the research participant, those individuals who decide whether they want their data in the dataset that the curator has. The last is the data analyst who submits queries to or who uses an anonymized dataset released by the data curator and who is assumed to be untrusted and malicious. It could be that the analyst is not malicious, but the model holds for the worst case.

The guarantee is a promise from the data holder to the data subject: “you will not be affected, adversely or otherwise, by allowing your data to be used in any study, no matter what other studies, data sets, or information from other sources is available” [9]. So the probability of harm due the participant in the study will not greatly change with participation or non-participation. It also does not create privacy where it was not before, as participating in a dataset does not make an individual more private. Also note that this does not mean any individual will not be affected, adversely or positively, from the conclusions drawn after releasing the data, as any such affect would be created whether the individual participated or not. Conclusions, or having datasets that teach, are after all why studies are done. This means that the same results can be learned from the resulting dataset whether or not a person participates in the dataset, because the probability of any sequence of outputs is essentially equal, so participation in the dataset will not be the reason for a breach of privacy. Note also that the participant does not have control over the rest of the dataset; the definition releases any such need [29].

Thus it protects the privacy of an individual while releasing useful information about a population [29]. For example, a campus Wi-Fi trajectory database could indicate that residence of a freshman dorm tend to walk towards the local bars on Thursday evenings. A negative affect for a freshman could be that the town sends out more police officers on Thursdays to arrest underage drinking, to the seeming detriment of the students. Though a positive affect could be that crime and negative health effects are reduced due to less underage drinking. It is certainly the case that more is known about the freshman in that dorm, but differential privacy argues that their privacy was not breached since it was the conclusion of the study that affected the freshman, not his/her participation in the study [29].

A privacy breach in this case would be learning someone participated in a database and thus being able to learn some sensitive information specifically about them. Note that this does not protect the privacy of individuals whose data was not collected, meaning as stated above it does not provide privacy where there is none. This means that differential privacy protects privacy beyond re-identification. Re-identification is a real risk, as previous algorithms have attempted to protect against, but privacy can still be breached by other methods [29]. For example, an attacker could narrow down an individual's record to a small subset of the de-identified dataset, meaning they do not know exactly which records is the individual's but they learn attributes this individual could have with high probability.

Dwork also shows that other seemingly private notions are not actually private [29]. Queries over large datasets are not private because of attacks such as differencing attacks which can be constructed with multiple queries to reveal information. Also, preventing queries because they may reveal information reveals that that query is sensitive. Summary statistics can be broken with differencing attacks, reconstructing the dataset from the statistics, and others. Releasing seemingly inconsequential data could reveal more than intended. These ideas, among those described previously and others, form a motivation for using differential privacy.

This definition gives a reliable way to distribute sensitive information for the benefit of research and societal gain [29]. It does not, though provide any absolute guarantee of privacy since any dataset with utility compromises some level of privacy [30], but bases the level of privacy on the algorithm itself.

Differential privacy is in the randomization-based scheme category since it perturbs data [25]. There have been other randomization schemes [31], but differential privacy does not have to be implemented through noise addition and as stated simply provides a definition from which mechanisms can be built. It can also be classified as semantic anonymization since it relies on the logical relationship between data. It does not go as far as semantic security in the cryptography setting which says that nothing is learned from a plaintext, since as stated earlier, the data analyst still needs to learn and the analyst and adversary are thought of as the same person, but it provides semantic guarantees nonetheless [29].

The strongest form of differential privacy is ε-differential privacy where ε is the bound on privacy. Namely, the absolute value of the privacy loss is bounded by ε. A relaxation is (ε, δ)-differential privacy which allows privacy breaches to occur based off a small probability δ, where the privacy loss is bounded by ε with at least probability $1- \delta$ [29]. More on the difference will be discussed in Section 0.

Because the privacy guarantee is indifferent to background information, it holds even if the attacker knows every other database entry. Thus it protects against very knowledgeable attackers.

It does seem as though some of the momentum for differential privacy died down after it was introduced, but it is conjectured that this is for two reasons. First, it is not a panacea for all anonymization needs, so improper usage where utility is not maintained brought negative light on it. Second, Microsoft placed many patents on key algorithmic features, thus reducing some endeavors. Yet there are real life usages of differential privacy. The U.S. Census Bureau has used it, specifically in the OnTheMap application [32]. This shows information about where workers are employed and where they live, but the data is synthetic data created with differentially private methods [33]. Google implemented it in their Chrome web browser as RAPPOR (Randomized Aggregatable Privacy-Preserving Ordinal Response) to anonymously report usage statistics of Chrome users [34]. This uses a coin flip technique to achieve the definition. Apple iOS 10 uses it in its intelligent assistance and suggestions, which affects service such as the keyboard, spotlight search, and Notes [35].

Despite its many benefits, like every algorithm, differential privacy does have its drawbacks. The next section will address its main criticisms relevant to the current research endeavor.

2.3.1 Criticisms of Differential Privacy

It seems appropriate at this point to address criticisms of differential privacy, both negative and positive. Since its inception, there have been various papers pointing out both underlying assumptions that must be addressed when implementing differential privacy and seemingly inherent drawbacks of the definition. This is not a full analysis nor defense of differential privacy, for that would be another thesis, but provides a general understanding of pertinent aspects of differential privacy that must be considered when applying it. Note that differential privacy will be discussed in more detail in Section 3.1.

A few years ago Bambauer et al. published a critique paper addressing limitations of differential privacy [36]. McSherry, an author of many seminal papers on differential privacy, responded in a quite virulent manner to this paper with a blog post on GitHub [37], and the authors replied with a rebuttal to the rebuttal [38]. The primary conclusion that can be taken from this back-and-forth is that differential privacy is not suitable for all applications. It will not provide accurate results when used on small populations and with overly specific questions. If functions with very large global sensitivities are chosen, the amount of noise that needs to be added will greatly decrease utility. Though relaxed forms of differential privacy do exist, i.e., (c, δ)-differential privacy, this negates the strong guarantees since even with a small δ there is no bound on privacy loss [39]. These relaxed forms could be useful but would have no more of a guarantee of privacy than other well-established SDC algorithms. Furthermore, though the Laplace mechanism criticized in these papers is the simplest mechanism that guarantees differential privacy, there are other mechanisms that provide better utility, especially if non-numeric aspects of a dataset are being used for the algorithm.

Some of the authors, namely Sarathy and Muralidhar, who wrote the paper just mentioned, made a few claims focused on the fact that the Laplacian mechanisms “does not satisfy the requirements of differential privacy” [40]. This though was based on an improper usage of sensitivity [37]. Namely they used local sensitivity, instead of global sensitivity, when adding noise, which was never actually used in differentially private algorithms since it was proven to not be differentially private [41]. They follow this up with two further papers [42], [43], which generally give similar arguments as already discussed about the drawbacks of using the Laplace mechanism for all cases. Though, as will be discussed in the next section, there have been many papers published that produce high utility results using differential privacy with various mechanisms.

Perhaps more important are a few other papers. Kifer and Machanavajjhala authored a paper clearing up some misconceptions about differential privacy [44]. Their main conclusion is that it does require assumptions about the data, namely how the data was generated. This is based on the claim of differential privacy that it protects against the knowledge of participation in a dataset. In order for differential privacy to be used, the individual records must be independent, i.e., the participation must be independent. Otherwise, the edges caused by dependency on other nodes or correlation with background knowledge may leak some information. This means that if the records are in fact dependent, there is a risk of privacy breaches from aggregate background information that has already been released. This consideration is addressed in Section 4.1 with relation to the specific data application chosen for this thesis.

Haeberlen et al., indicate that in the implementation of two differentially private systems, PINQ and Airvat, covert channel attacks are possible that could leak information [45]. These attacks were for interactive implementations of differential privacy so they do not apply in the non-interactive setting used in this thesis, but do show that implementations must consider other more practical attacks than just against the algorithm itself.

There has been some research into correctly choosing the value for ε. For example, Hsu et al., created a model which incorporates the privacy level desired by participants and fits the monetary budget and utility factor desired by the data analyst [46]. The main lesson in general is that ε values that are too small lead to low utility, while ε values that are too big do not give good privacy guarantees. This is because if it is small, the probabilities of the existence of an entry must be very close meaning large noise is added, while if it is large the probabilities can be so different that there is a very high chance that if a person's record is in the database it will be released and if it is not it will not be released (see Section 3.1 for the formal definition). More about which parameters to choose for the algorithm proposed in this thesis is in Section 6.2.

Because of these shortcomings and those described for syntactic based anonymization schemes, it should be clear that there is not one algorithm that provides a panacea for all anonymization needs. Yet despite the many aspects of differential privacy that must be considered while implementing such an algorithm, it still provides a strong definition with strong privacy guarantees. It provides protection against arbitrary risk, automatic neutralization of linkage attacks, a quantification of privacy loss, and immunity to post processing, among others. The next section will describe differential privacy as applied to the specific area of trajectory publication.

2.4 Privacy Preserving Trajectory Data Publication

As discussed in the previous section, partition-based schemes do not provide sufficient privacy protection, especially for trajectory publication due to the high dimensionality of that type of data [47]. Thus differential privacy is chosen because it provides a strong privacy guarantee. This thesis specifically studies the non-interactive setting of differential privacy with respect to Privacy Preserving Trajectory Data Publication (PPTDP). The two setting of privacy algorithms are interactive and non-interactive. In the interactive setting, a trusted data collector provides an interface to which users can submit queries to receive anonymized results. In the non-interactive setting, the trusted data collector publishes a full dataset that has been anonymized, and thus users can query this resulting dataset as much as desired. Dwork focused on the interactive setting, but it has since been applied to the non-interactive setting by many authors. When the types of queries are known, the non-interactive setting can give better accuracy since all the noise is added based on the expected queries, whereas in the interactive setting, noise has to be added with each query [29]. The non-interactive setting is generally preferred by statisticians, so that is the setting used [48].

Anonymizing trajectories is not a new field of study, but many such algorithms use the partition-based privacy modes which, as noted previously, do not guarantee a verifiable level of privacy. In the work that has been done in the past, there are generally two different methods for publishing trajectories [49], [50]. The first puts all the trajectories in a database and anonymizes the trajectories each as an individual record. The second treats each trajectory as a database and anonymizes the locations within each trajectory.

Whichever method is chosen, differential privacy, introduced by Dwork, provides a formal guarantee of a verifiable level of privacy [28]. These differentially private algorithms work by aggregating some sort of statistic or value from the raw dataset and constructing synthetic sequences by perturbing these raw numbers based on a differentially private mechanism. The most common mechanism is the Laplace mechanism which is used to add noise drawn from the Laplace distribution to the raw data [48]. Also, in order to fulfill differential privacy, every possible location in the location universe must be considered when constructing the synthetic trajectory since every possible trajectory has to have at least some probability of appearing in the resulting dataset [47]. Though, various algorithms implement strategies to fulfill this requirement in an efficient manner, as will be described below.

Chen et al., were one of the first to apply differential privacy to PPTDP [47], [51]. In the first, they invoke a two-step process where they use the raw trajectory dataset to create a prefix tree based on the noisy count of each node occurring in a prefix and then generate the synthetic database [47]. They later applied this to publishing data from the Montreal transportation system [19]. A prefix tree is a tree structure holding the locations of trajectories up to a specified tree height. Using this tree structure narrows down the output domain from considering all possible trajectories to those possible in each branch. They iteratively create this tree level by level, adding Laplacian noise to true counts of trajectory prefixes and continue expanding a node if the count passes a threshold θ. Note they uniformly divide the privacy budget ε between each level of the tree. Trajectories that are longer than the tree height are truncated, which does not unduly affect utility since there will be few trajectories that are of that length that are the same. Then two constraints are imposed, namely that each child count has to be less than or equal to parent counts, and the sum of children counts has to be less than or equal to the parent count. This is done by using estimations using the minimum L2 solution (the L2 norm is a vector norm, or length, of part of the trajectory) to change the noisy counts. Lastly the synthetic trajectories are created by traversing the prefix tree in postorder, from the height to the root. This is a relatively simple algorithm but does not generally scale well and provides less utility than later works since with a larger prefix tree the number of prefixes in a branch decreases [51].

In the second paper by Chen et al., they used a variable length n-gram model to achieve ε-differential privacy [51]. They tested it on MSNBC website visit data and STM (the transit system in Montreal) transportation station visit data. This takes as input a trajectory database, maximum n-gram size, maximum trajectory length, and ε. The first step is to create counts of n-grams from the raw dataset up to a maximum size then interactively creating an exploration tree by adding all possible children to each node and adding Laplacian noise to get noisy n-gram counts. Next, consistency constraints are enforced, editing noisy counts based on predictions from a Markov assumption. Lastly, the exploration tree is extended to the maximum trajectory length and synthetic trajectories created in post order from the tree height to its root. This algorithm uses the Laplace mechanism although other noise generation mechanisms could be used instead which could lead to more utility. The n-gram counts were relatively large, resulting in lower Laplace error, meaning lower amounts of noise added. This also slightly raised approximation error, but nonetheless essential sequential information is maintained. They proved it surpassed the prefix tree structure just discussed based on count query and frequent sequential pattern mining utility results. Yet due to the need for large n-gram counts, it is possible that this algorithm will not produce high-utility results for some datasets.

He et al., recently presented Differentially Private Trajectories (DPT) which uses a hierarchical reference system to represent sampled trajectories [52]. The example they use is taxis driving around a city. This algorithm aims to capture the changing speeds of objects creating trajectories, i.e., taxis driving in a city, while creating a more efficient algorithm for trajectory publication in large spatial domains. DPT takes the spatial domain and discretizes it into different reference systems of different resolutions, thus the hierarchical reference system. Parts of each raw sequence is stored in a prefix tree corresponding to different reference systems based on trajectory speed of that segment. Laplacian noise is added to these prefix trees and they are adaptively pruned out and their heights modified based on utility requirements, namely to ensure only transitions that fit into the overall hierarchical reference system remain. Finally, the resulting sequences are sent through a direction weighted sampling algorithm based on a subset of the previous transitions to ensure inherent directionality found in real trajectories is present in the synthetic trajectories. This whole algorithm reduces the amount of nodes considered as children when constructing synthetic trajectories to the eight cells around the current location plus possible movement to another reference system, thus increasing efficiency especially over Chen et al.'s systems which have to consider every possible location in the universe. Yet, this paper also only looks at sequential data, i.e., time-ordered data, and not trajectory data, i.e., time-location pairs. This algorithm provides more utility and is faster than Chen et al.'s n-gram solution, but would also require knowing the exact location of each AP, which currently is not available.

There are also papers that take a slightly different approach. In 2013 Jiang et al., proposed a method for ensuring the noise added produce realistic trajectories [49]. Instead of using the basic Laplace mechanism, they use the exponential mechanism [53]. So instead of choosing the next location based on adding proportional noise, a location is chosen from the universe that has probability exponentially proportional to a quality score. This generally means the resulting synthetic data has more utility than data created with the Laplacian mechanism. In this case the quality score is the closeness of the point to the trajectory's terminal point, meaning each consecutive location has to progress towards that point based on the result of a formula. To publish trajectories, their mechanism, Sampling Distance and Direction (SDD), calculates the norm and direction of a vector between a starting point and the next point, using the exponential mechanism to compute a norm and angle that is sufficiently close to the real norm and angle based on the quality score described above. This method could work well if the actual geographical locations of APs is known, but again that is not the case. It also focuses on publishing trajectories with known starting and ending points instead of a full database of trajectories.

There are a few papers that handle trajectories instead of sequences. Shao et al. used interpolation and sampling instead of noise addition to produce an (ε, δ)-differentially private algorithm [50], the slight relaxation on differential privacy, though their paper is quite short on details. They propose two different algorithms, focusing on publishing ship trajectories. The first is an a priori sampling mechanism which randomly keeps a location in each trajectory then interpolates the removed positions based on cubic Bézier interpolation. The random position is in the range 1 to k where k is the inverse of δ. The second is an a posteriori sampling mechanism which works in the opposite order. First it interpolates positions of a trajectory based on cubic Bézier interpolation then it samples this interpolation to generate a new trajectory. This algorithm does handle trajectory data, but it focuses on publishing individual trajectories instead of full trajectory databases and uses a weaker version of differential privacy.

Another such paper is that by Hua et al. in which they use generalization to better represent actual time-series trajectories [54]. It uses two ε-differentially private algorithms to achieve this and is tested with taxi movement data. The first algorithm generalizes all the locations into groups at each timestamp with k-means clustering based off pairwise Euclidian distance, meaning double the amount of groups are formed since the distances are calculated in pairs. Then the exponential mechanism is used to modify the members of each group to contain the trajectories which have the least utility loss. So the utility function used for the exponential mechanism in this case is based on the Euclidean distances. Instead of considering every possible trajectory that could be drawn from this generalized location universe, the second algorithm publishes new trajectories by adding Laplacian noise to the generalized trajectory counts, these are in the original database, and then iteratively adds new trajectories with random Laplacian counts. To add these random trajectories, it sorts the original trajectories based on noisy counts and picks a set number of random trajectories within each noisy count interval based on a probability of trajectories having a count within that range. Synthetic trajectories are added until the total number is equal to the number of trajectories in the input dataset. This algorithm handles trajectory data, time-stamped sequences, and works well with sparse datasets, or datasets where trajectories are generally unique partly because of a high location universe, but it again it would require the knowledge of actual locations of APs.

Last year Li et al. created an algorithm that also handles trajectories and focuses on protecting the privacy of sensitive geographical areas since trajectories created with unbounded Laplacian noise could create abnormal trajectories [55]. This is a modification of Hua et al.'s algorithm with the difference in the publishing step. Only a portion of the generalized trajectories are picked, those with the highest n_1 counts, and then the remaining trajectories are also picked from the generalized trajectories. Also, if the generated noisy count goes above a threshold, that trajectory is not added. Laplacian noise is still added and the algorithm still stops when the number of synthetic trajectories reaches the raw trajectory count. This algorithm slightly improves efficiency from Hua et al.'s work but also suffers from the same drawback.

These were all differentially private PPTDP algorithms with their basic inner workings and criticisms based on the application sought. Next will be discussed privacy preserving software available to data holders.

2.5 Software

There has been a development of a small range of anonymization software [56]. It would have been desirable to use one of them, but despite a seemingly diverse array of options, very few were found that anonymize trajectory data and use differential privacy. For this reason, a general overview of such software is given, starting with such software that focuses on partition-based schemes.

ARX provides the most extensive framework that allows integration of many different algorithms while also providing a clean interface [56], [57]. All of the pieces of software found base their algorithms on partition based anonymity. ARX may be the exception because it does claim to use differential privacy, but their version is based off of k-anonymity and uses (ε, δ)-differential privacy. The other problem with these is they are focused on tabular data, and not trajectory data so a restructuring of the data handling would be required. Data in ARX is represented in an integer array, generalization hierarchy, and dictionary in order to efficiently correspond the input data to varying levels of generalization. This means fundamentally the data is viewed as a database of rows and columns, which does not lend itself to anonymizing trajectories with trees. Also, their Flash Algorithm is based on finding an optimum anonymization with minimal information loss for partition based schemes.

Other pieces of anonymization software that primarily use partition based schemes are as follows. The UTD Anonymization Toolkit provides an efficient framework for large datasets but has no attacker analysis or GUI [58]. The Cornell Anonymization Toolkit provides good documentation on its structure, but does not quite provide as many features as ARX [59], [60]. TIAMAT is a Java based k-anonymity tool, though the code could not be easily found [61]. SECRETA is a C++ tool but there was no API [62], [63]. Open Anonymizer is a very simple tool written in Java and Javascript [64], [65]. AnonTool is a tool specializing in sanitizing medical data but again the code could not be easily found [66]. µArgus was created by the Netherlands government and has a very detailed manual, but no API for easy integration [67]. sdcMicro is an improvement on µArgus with a GUI and open API [68] . There are also a limited amount of commercial products including those by Privacy Analytics [69].

There are a few pieces of software that focus on anonymizing trajectory data, most notably VisDPT [70]. This is a GUI form of the DPT algorithm described earlier by He et al. It provides an implementation of DPT with privacy and utility analysis and visual plotting of trajectories. Though it does suffer from the same shortcomings as DPT in regards to the data limitations in this research.

2.6 Conclusions

There have been many algorithms used for data anonymization. Recently, differential privacy has come out as a top contender. Though there are considerations that must be addressed when implementing a differentially private algorithm, it is chosen as the base for anonymizing the trajectory data in this thesis. With the benefits and shortcomings of each PPTDP algorithm, Chen et al.'s n-gram algorithm provides the middle ground between ease of implementation and ease of use for the focus of authmgr data. What is more, they claim it is relatively straight forward to add time information by labeling each node with a location and timestamp, instead of just a location. Note that He warns against using previous prefix tree (or n-gram tree) approaches since the discretization of a large spatial domain would result in a large number of possible locations, but in the case of APs on a campus it is assumed this discretization into AP locations would be sufficiently small to use a similar approach. This assumption will be addresses in further in Chapter 6. The next chapter will describe pertinent background knowledge needed for the algorithm design.

3 Background

As the related work has been discussed, this section will address pertinent algorithmic concepts. First it will further explain differential privacy, its definition, implementation details, and criticisms. Differential privacy is a definition of privacy that provides formal guarantees related to privacy loss. Then it will discuss the definition and usage of the n-gram model which is pertinent to the algorithm chosen.

3.1 Differential Privacy

The basis of differential privacy having been discussed, its formal definitions follows. For the guarantee of differential privacy discussed in Section 2.3 to be true, the ability of an adversary to inflict harm on any set of people should be the same whether any individual participates or does not participate in the dataset. Thus differential privacy handles all forms of good and bad outcomes by studying how the probability of a privacy mechanism can change with the addition or deletion of any row (or in this thesis' case, any trajectory) from a dataset. So this definition is based on the closeness of probabilities of neighboring databases, which are those that differ in one record. This means one is a subset of the other [9]. The following is a formal definition of differential privacy.

Definition 3.1. A randomized function that maps a database to real numbers, $f: D \rightarrow \mathbb{R}$, gives ε-differential privacy if for any databases D_1, D_2 differing in at most one record, and for any possible output $O \subseteq Range(f)$,

$$Pr[f(D_1) \in O] \leq e^{\varepsilon} * Pr[f(D_2) \in O],$$

where the probability is taken over the randomness of f [28].

This also implies a lower bound [46]

$$e^{-\varepsilon} * Pr[f(D_2) \in O] \leq Pr[f(D_1) \in O] \leq e^{\varepsilon} * Pr[f(D_2) \in O]$$

This means that the probabilities are bounded in both directions so that the output probability on D_2 is at least $e^{-\varepsilon}$ times the probability of an output in D_2 and at most e^{ε} times that probability.

Another way of looking at this definition is moving the probabilities to one side and creating the following inequality [71]

$$e^{-\varepsilon} \leq \frac{Pr[f(D_1) \in O]}{Pr[f(D_2) \in O]} \leq e^{\varepsilon}$$

This is a clearer representation of the bound that ε places on the output probabilities, showing how the harm is almost the same regardless of participation since these two probabilities must be similar in order to fulfill this inequality. The harm, represented by the differences in probabilities, is bounded by a lower and upper bound. For example, if the probability of the function on D_1 having the output O is 56.12% and 56.13% for D_2, the fraction of these probabilities would be 0.9998. If $\varepsilon = 0.1$, then the above inequality would become: $0.9900 \leq 0.9998 \leq 1.010$, which is true. Even if the probabilities are switched, the inequality would still hold, $0.9900 \leq 1.000 \leq 1.010$, so it does not matter whether D_1 or D_2 is the smaller database, just that they differ by one entry. Also from this inequality comes a notion of privacy loss

$$\ln\left(\frac{Pr[f(D_1) \in O]}{Pr[f(D_2) \in O]}\right)$$

Note that this value could be positive or negative. It follows the same logic as described for the above inequality. Namely, the absolute value of this natural log value is the ε value, or the amount of privacy loss.

The parameter ε measures the level of privacy while the exact level of utility must be measured by other means. In general, larger ε means less privacy but higher utility, while smaller ε means the opposite. Also, as an individual participates in more research studies that are differentially private, their privacy risk will increase with each ε value [9].

At this point the difference between ε-differential privacy and (ε, δ)-differential privacy will be briefly discussed. ε-differential privacy guarantees that with every run, the privacy mechanism will output with almost equal probability the same output for neighboring databases. (ε, δ)-differential privacy on the other hand dictates that the output of the mechanism is unlikely to be different for neighboring databases, but it could be found that one is more likely to produce the output. Again, because the second definition is weaker, this thesis follows the concept of ε-differential privacy.

Formal definitions related to differential privacy have been presented and discussed. With this foundation, specific implementation details will be discussed.

3.1.1 Implementing Differential Privacy

Differential privacy is implemented with privacy mechanisms, or mechanisms [29]. In the non-interactive setting as chosen in Section 2.4, this takes as input a dataset, set of all possible data types (the universe which in the trajectory case would be the set of all possible locations), and a random number, and outputs a synthetic dataset. Note that in Definition 3.1, f contains the privacy mechanism, i.e., f uses the mechanism along with other algorithmic content to achieve differential privacy.

An important concept when applying mechanisms is the global sensitivity as defined as follows:

Definition 3.2. For a function $f: D \rightarrow \mathbb{R}^d$, the global sensitivity of f is

$$\Delta f = \max_{D_1, D_2} \| f(D_1) - f(D_2) \|_1$$

$$= \max_{D_1, D_2} \sum_{i=1}^{d} |f(D_1)_i - f(D_2)_i|$$

for D_1, D_2 all differing in at most one element [28].

This is also called the L_1 sensitivity since it uses the L_1 norm. It describes the maximum amount of impact the addition or removal of an entry can have on the output of the chosen function, f. Differential privacy works best when global sensitivity is small since the least amount of noise has to be added with smaller sensitivities. It should be noted that this concept of sensitivity is a property of the function and not of the database, so it is independent of the actual raw data. For example, for a function f that is a counting query which counts the number of Android users in a network, any user can only change this number by one so the sensitivity of f is $\Delta f = 1$. The sensitivity of the function used in this algorithm is described in Section 5.3.3.

There are a few different ways to implement differential privacy. Usually, a mechanism is used that fulfills the definition. One of the simplest and first introduced mechanisms is the Laplace mechanism. The concept is that noise is randomly added from a distribution centered around the actual value from the raw dataset. Thus, the noisy result is a random value that is close to the original value. The "closeness" of this is based on two factors: the global sensitivity and privacy budget ε. The general formula for noise addition is

$$X' = X + Lap(\Delta f / \varepsilon)$$

where X is the real answer and X' is the noisy answer. The Lap, or $Lap(\lambda)$, term is used to denote a random variable drawn from the Laplace distribution, which will be discussed below. Note again that the noise addition depends inversely on ε and directly on Δf. Thus with larger sensitivities and smaller budgets, more noise is added. The converse is also true.

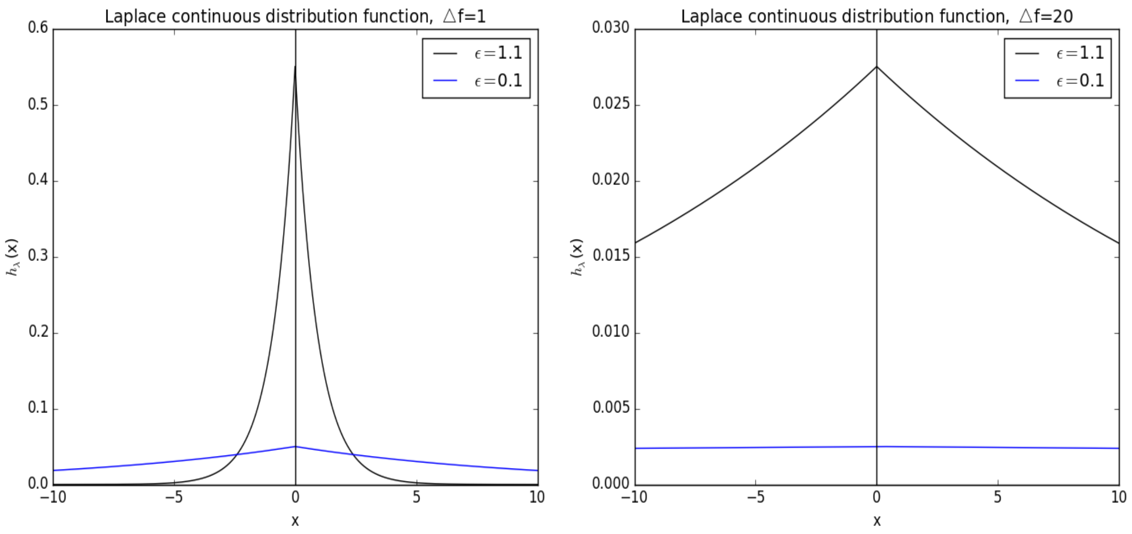


Figure 3.1 Laplace CDF for different sensitivities and ε values

Figure 3.1 shows the cumulative distribution function (CDF) for the Laplace distribution at different sensitivities and ε values. The CDF shows the probability that the function will return a value less than or equal to the input. Thus with smaller ε values or larger sensitivities, this probability is smaller, meaning the value returned and thus noise is larger.

In order to use the Laplace mechanism, a random variable, Y, is drawn from the distribution. This is represented by the $Lap(\lambda)$ term above and can be done with taking a single draw from the distribution. The following equation is based on taking the inverse of the CDF. U is a random variable drawn uniformly from $(0,1)$ [72]

$$Y \leftarrow sign(r) * \lambda \ln(1 - 2|r|)\,, where\ r \leftarrow U - 0.5$$

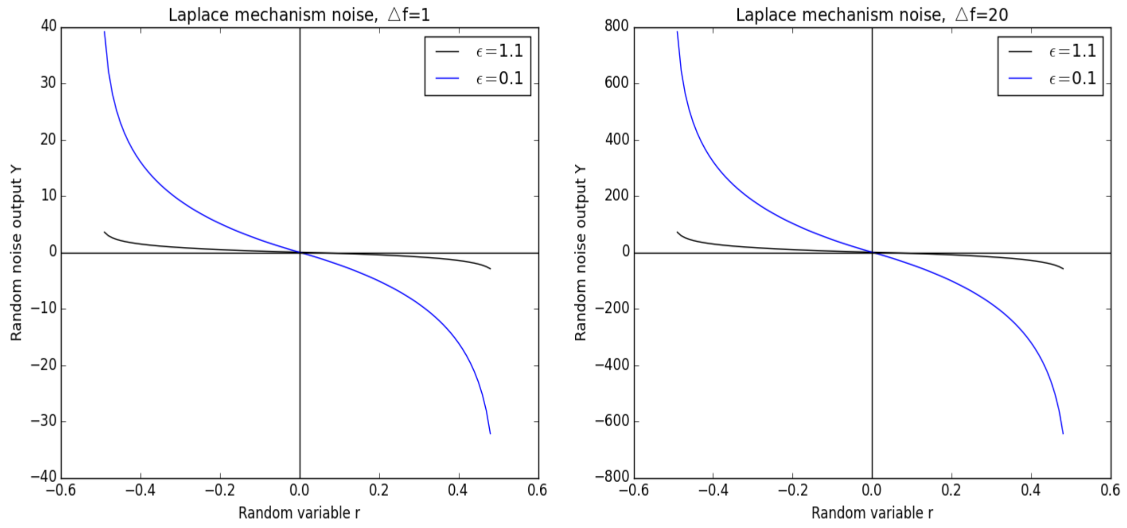


Figure 3.2 Noise distribution from Laplace mechanism for different sensitivities and ε values

Figure 3.2 shows the magnitude of noise added for any r value randomly chosen within this -0.5 to 0.5 range. Again, with smaller ε values or larger sensitivities larger amounts of noise are added.

There are other mechanisms that can be used as well. The Gaussian mechanism can provide a way to add less noise since the distribution is more tightly coupled around 0, but only upholds (ε, δ)-differential privacy [30]. It also uses the L_2 sensitivity (the L_2 norm minimizes the sum of the squares of the differences, and not the absolute differences) [29]. Noise addition is also not the only way to achieve differential privacy. The exponential mechanism is useful for functions that do not produce numeric results, for example mapping a database to some sort of string [53]. It returns a weighted sampling from the collection of all the possible outputs [17], [54]. There has also been work in using random sampling combined with k-anonymity to achieve differential privacy though this only achieves the relaxed δ form but actually does add random noise following the Laplacian distribution to true statistics of the original data [73].

There are also a few important properties when creating differentially private algorithms. First, there are two composition properties, sequential and parallel [54], [74]. An ε_1-differentially private algorithm combined in sequence with an ε_2-differentially private algorithm produces an $(\varepsilon_1 + \varepsilon_2)$-differential privacy result. On the other hand, applying an ε_1-differentially private and ε_2-differentially privacy algorithm in parallel on two disjoint datasets produces a $max(\varepsilon_1 + \varepsilon_2)$-differential privacy result. This also applies to ε_1-differentially private functions within an overall algorithm. Thus the total ε budget can be split up between functions and parts of the data processing. Second, it is also immune to post-processing [29]. When designing an algorithm this means that after adding Laplacian noise or performing the differentially private functions, post-processing can be done on the data to make it more accurate as long as this is done with the sanitized statistics and not raw statistics [75].

Global sensitivity, privacy mechanisms, and composition properties are all essential concepts when implementing differential privacy. In the next section, some drawbacks of differential privacy will be revisited.

3.1.2 Drawbacks of Differential Privacy

After discussing the formal definitions related to differential privacy and implementation details, some of the criticisms discussed in Section 2.3.1 will now be discussed. These criticisms relate to choosing the privacy budget, dataset size, groups, and implementation issues.

The largest issue related to parameter selection is choosing ε. It is relatively easy to measure utility of an algorithm implementing differential privacy, as is done in Chapter 6. The caveat here is that this utility level is rather data dependent. Thus, the utility results of such algorithms must be considered in relation to the specific data used for testing. The problem is choosing the ε value for the desired level of privacy. It is clear that a smaller value guarantees more privacy, while the opposite is true, but thus far there has been no actual privacy quantification related to specific values of ε. This thesis studies how different ε values affect utility and not necessarily privacy, since with all values of ε, at least relatively small ones, differential privacy still holds guarantees [29].

An issue related to data is that differential privacy is most useful with large datasets since adding noise or achieving the guarantees in another form adds proportionally more noise to small amounts of data. It was assumed that there was enough authmgr data, but as seen in Chapter 6 it is clear that the specific statistics taken from the data make the large dataset almost act like small datasets when adding noise.

It also does not protect privacy of groups. The solution to this is described in Section 3.1.2 related to the independence of data, but here is presented a possible alternate solution. There is a definition called group differential privacy which is defined as:

Definition 3.3. A randomized function that maps a database to real numbers, $f: D \rightarrow \mathbb{R}$, gives kε-differential privacy for groups of k if for any databases D_1, D_2 differing in at most one record, and for any possible output $O \subseteq Range(f)$,

$$Pr[f(D_1) \in O] \leq e^{k\varepsilon} * Pr[f(D_2) \in O],$$

where the probability is taken over the randomness of f [29].

As stated, the k indicated the number of members of each group. Thus, instead of protecting the privacy of the participation of one individual, it protects the privacy of a group of k individuals. The issue with this is that the group size must be known and would be capped at k.

There are also implementation issues as described earlier. These will be addressed in the next chapter, the Implementation chapter. The basic differential privacy definition protects any individual from privacy loss and is usually implemented via the Laplace mechanism. These concepts can be integrated into different algorithms, including the n-gram model

3.2 N-gram Model

An n-gram model is a probabilistic prediction model based on the (n-1)-order Markov model making use of the fact that the probability of the occurrence of an item in a sequence is only based off the past n-1 items [51]. Thus it is a compact way of representing a large sequential database, losing some accuracy but also reducing storage needs. It has also been proven to be useful in representing sequential information [51].

It is based on the Markov independent assumption of order n-1, which states that the occurrence of each item in a sequences depends only on the previous n-1 items, and not on all previous items [51].

This process is important for the approach taken in the following section. The n-gram model fits nicely into differentially private algorithms since it can retain high quality information about the raw dataset and provides a means for adding noise, i.e., to the n-gram counts. This is true as long as the n-gram counts are large enough to not be overly affected by Laplacian noise.

3.3 Conclusions

Differential privacy provides a privacy definition with strong mathematical guarantees. Integrating a variable length n-gram model with differentially private mechanisms provides a means for retaining utility of the dataset while adding noise to guarantee the differentially private guarantees. The next chapter will discuss the base algorithm used, then the following chapter after that will discuss specific changes to this algorithm in implementation.

4 Algorithm

To fulfill the aim of this thesis to discover if differential privacy (dp) can be used to anonymize network data from a large university wireless network, a framework around Chen, Acs, and Castelluccia's algorithm is created for processing authmgr data, as defined in Section 1.3. This system contains a few different sections. The first is to extract the authmgr logs for the IT storage container and create trajectories from these logs. The second is to implement the anonymization algorithm that takes as input these raw logs and outputs synthetic authmgr logs. As stated in Section 2.6, the algorithm used for this step, referred to as ngram2.0, is a slightly edited version of Chen et al.'s n-gram algorithm, referred to as ngram. The third and last step is to perform utility functions on the input and output databases to ensure utility is preserved. Anonymity is guaranteed by differential privacy.

Chen et al.'s algorithm will further be referred to as ngram. Their Python implementation, presumably by the author Acs, is referred to as ngrampy. This is referred to by a different name as what is described in the paper since they implemented the concepts in a slightly different manner than following ngram word by word. The algorithm used by this thesis is named ngram2.0 as it is an upgrade of ngram. Namely, ngram2.0 is extended to work with spatiotemporal data and provides slight implementation changes. This chapter describes the algorithm placed into the framework mentioned above. The aim is to describe the ngram2.0 algorithm. To do this, it describes the ngram algorithm then details the changes in ngram2.0. It describes the ngram algorithm since many of the main pieces are generally the same, with minor tweaks, although some major changes are discussed as well. Specific implementation details are then discussed in the next chapter.

The basic concept of the algorithm is that high quality n-grams are calculated from the raw database to which Laplacian noise is added. These n-grams are then released. The actual n-gram creation process is not discussed by Chen et al., but the algorithm that creates synthetic n-grams by adding noise to a n-gram tree structure is discussed. There are three main phases. The first is Exploration Tree Creation (Section 4.4), which creates a tree structure to hold resulting n-grams while adding noise to the raw n-gram statistics. This phase is the largest and contains many steps to fulfill its goal. The second is Enforce Consistency Constraints (Section 4.5), which takes the resulting synthetic exploration tree and slightly changes noisy n-gram counts to create an exploration tree with features that resemble an exploration tree created from raw n-grams. The third and final phase is Construct Synthetic Database (Section 4.6), which creates a synthetic sequential database from the resulting exploration tree created in the previous two steps.

The algorithm implemented in this chapter, ngram2.0, has a few major changes to previous work, namely ngram. There are three main differences. First, the framework, mentioned above, is implemented to allow for future algorithms to be implemented in the same code base. Second, the algorithm handles spatiotemporal data instead of just spatial data, which adds minor yet significant dimensional features, as discussed in Section 1.3. Third, the input dataset can be in the form of a tabular or sequential dataset. There are also a number of minor contributions. Through all the phases, the algorithm is described in a more programmatic way, instead of the more theoretical ngram description. There is a new Preprocessing phase. The Exploration Tree Creation phase changes node state calculation to better handle state and rounds budget calculation to fix infinite calculation problems. The Construct Synthetic Database phase combines ngram and ngrampy methods to create a cleaner algorithm.

Timestamp	AP Name	Src Mac	Username
2017-07-07T13:59:00.048Z	LIB-234BB1034M	6c:94:f8:12:34:56	pid1@vt.edu
2017-07-07T13:59:00.048Z	LIB-234BB1034M	6c:94:f8:65:43:21	pid2@vt.edu
2017-07-07T13:59:00.048Z	LIB-234CA1075S	6c:94:f8:78:9a:bc	pid3@vt.edu
2017-07-07T13:59:00.048Z	LIB-234BB1034M	60:67:20:12:34:56	pid4@vt.edu
2017-07-07T13:59:00.048Z	LIB-234CA1075S	60:67:20:78:9a:bc	pid5@vt.edu
2017-07-07T13:59:30.048Z	LIB-234BB1034M	60:67:20:de:f1:23	pid6@vt.edu
2017-07-07T13:59:30.048Z	LIB-234CA1075S	3C:97:0E:12:34:56	pid7@vt.edu
2017-07-07T13:59:30.048Z	LIB-234CA1075S	3C:97:0E:78:9a:bc	pid8@vt.edu
2017-07-07T13:59:30.048Z	LIB-234CA1075S	6c:94:f8:12:34:56	pid1@vt.edu
2017-07-07T13:59:30.048Z	LIB-234CA1075S	6c:94:f8:65:43:21	pid2@vt.edu
2017-07-07T13:59:30.048Z	LIB-234BB1034M	6c:94:f8:78:9a:bc	pid3@vt.edu
2017-07-07T13:59:30.048Z	LIB-234BB1034M	60:67:20:78:9a:bc	pid5@vt.edu
2017-07-07T14:00:00.048Z	LIB-234CA1241A	6c:94:f8:12:34:56	pid1@vt.edu
2017-07-07T14:00:00.048Z	LIB-234CA1075S	60:67:20:12:34:56	pid4@vt.edu
2017-07-07T14:00:00.048Z	LIB-234CA1241A	60:67:20:78:9a:bc	pid5@vt.edu
2017-07-07T14:00:00.048Z	LIB-234CA1075S	60:67:20:de:f1:23	pid6@vt.edu
2017-07-07T14:00:00.048Z	LIB-234CA1241A	3C:97:0E:78:9a:bc	pid8@vt.edu
2017-07-07T14:00:30.048Z	LIB-234CA1241A	60:67:20:12:34:56	pid4@vt.edu
2017-07-07T14:00:30.048Z	LIB-234CA1241A	60:67:20:de:f1:23	pid6@vt.edu
2017-07-07T14:00:30.048Z	LIB-234BB1034M	3C:97:0E:12:34:56	pid7@vt.edu
2017-07-07T14:00:30.048Z	LIB-234BB1034M	3C:97:0E:78:9a:bc	pid8@vt.edu
2017-07-07T14:01:00.048Z	LIB-234BB1034M	60:67:20:de:f1:23	pid6@vt.edu
2017-07-07T14:01:00.048Z	LIB-234CA1075S	3C:97:0E:78:9a:bc	pid8@vt.edu
2017-07-07T14:01:30.048Z	LIB-234CA1075S	60:67:20:de:f1:23	pid6@vt.edu

Table 4.1 Example 1 sample authmgr tabular data

Throughout this chapter, an example will be used to better illustrate the steps that are taken in this algorithm, referred to as Example 4.1. This example starts with a fake authentication manager (authmgr) dataset, which is shown in Table 4.1. Authmgr data will be discussed more fully in Section 6.1, but for now note that each entry or row contains four main fields. Each row is associated with a user (Username) and device (Src Mac) connecting with an access point in a wireless network (AP Name) at a certain time (Timestamp). Again all data used in this example is fake, although the actual AP Names are from the wireless network studied. The data from Table 4.1 was run through the ngram2.0 code to produce the output database and intermediate exploration tree values. Each phase of the algorithm has a section which shows how that phase affected this data in that particular run of ngram2.0. This is just an example of what the resulting intermediate phases and resulting datasets could look like.

Each Section of this chapter covers a phase of the ngram2.0 algorithm. Each section starts by describing what ngram does for that part of the algorithm, if it does anything, then described the ngram2.0 approach. Before describing these steps, basic terminology is discussed next, in Section 4.2.

4.1 Considerations

The focus of this implementation is for sparse data. In this particular application this data is wireless network log data, but could be any other sequential data with or without timestamps and identifiers, such as train stop user movement data. The sparsity measure of this data should be in accordance with the results found in Section **Error! Reference ource not found.**.

The first consideration stems from the definition of dp and the issue of independence discussed in Section 2.3.1. Again, data must be mutually independent for dp to be valid. In the specific case used for this thesis, the participation of a trajectory in the dataset means the presence of a device, associated with an individual, on campus. In order to be mutually independent, events must be pairwise independent, i.e., for each pair the probability of their intersection equals the product of the probabilities, and the probability of the intersection of all events equals the product of all the probabilities. As with the data in [19], it is assumed that the records in the authmgr data that are being used are independent because the presence of the trajectory of an individual is not dependent on the presence of the trajectory of other individual. The one situation when this could not be the case is when individuals have multiple devices. Thus multiple trajectories would be dependent on the same person being in the dataset. In this case, the concept of group dp described in Section 3.1 could rectify the situation. The problem becomes how big to create the group. Thus we assume all trajectories are independent. Also, if the algorithm proposed in this thesis is used, there is no plan to release deterministic statistics about the original dataset.

The next consideration pertains to the data characteristics of the authmgr dataset. The exact geographical location of the items in the universe of locations are also not known. If data such as latitude and longitude were used, the whereabouts of each location would be known and thus relational qualities between them, such as relative distance and absolute position, could be calculated. With this dataset, this precise location is not known for all points. Thus the algorithm cannot use spatial relational data to improve performance.

To produce a sanitized dataset that is most representative of the format of an input dataset, the overall characteristics of time, location, and identifiable information should be retained. The preceding implementation dealt with ensuring time and location are released in a private yet useful way while giving each trajectory a unique identifier. This method works if the identifying elements are viewed as one unique identifier for each event but there are cases where there are identifier patterns associated with these time/location pairs. For example, as mentioned before with authmgr data, each event has a device MAC address (a unique identifier that every internet connected device has) and personal ID associated with each spatiotemporal point. Each MAC/ID combination is unique, but each ID could have multiple MACs associated with it, i.e., a user has a smartphone, tablet, and laptop all signed into the wireless network under their ID. This would mean this ID is associated with three trajectories, while each MAC/ID combination is still only associated with one trajectory.

There are two problems in this situation. First, there are many anonymization schemes that are applied to tabular data, but most often specifically identifying information such as names are suppressed. Thus these studies do not deal with handling this type of data. Second, the algorithm that this thesis' implementation is based off of is specifically designed to handle spatiotemporal points without other fields. Noise is added to the count of $key(v)$, where v is a node at a time, location pair (per Table 4.2), irrespective of the identifiers associated with $key(v)$.

To rectify these problems, one approach could be defining the $key(v)$ as a time, location, identifier triplet, but this would add an even greater amount of overhead leading to even less scalability of the implementation. Another approach could be to use histograms or kd-trees as has been researched with partition based schemes such as t-closeness [21]. Again though the problem is associating these histograms with the trajectories in the output dataset. It is left to future work to combine the approaches of anonymizing trajectory and associated metadata together.

These considerations brought Chen's n-gram algorithm to the forefront as the ideal algorithm to be chosen, as Section 2.6 mentioned. This algorithm does not need exact geographical locations and can easily be extended to handle spatiotemporal data instead of straight spatial data.

The general data requirements and assumptions have now been addressed. The next section will lay down definitions of terminology used, before arriving at the first step of the algorithm, namely data preprocessing.

4.2 Terminology

Before discussing the algorithm, some terms need to be defined. For ease of lookup, Table 4.2 shows terminology used for the algorithm. The majority of these terms are from

Chen's paper, but some are newly defined or changed slightly for consistency. The table indicates the source for each term.

Symbol	Meaning	Source
$\mathcal{D}$	Input database in tabular form	+
$\mathcal{D}'$	Trajectory (or sequence) database	+
$\widetilde{\mathcal{D}}'$	Output (noisy) database in trajectory form	+
$\widetilde{\mathcal{D}}'$	Output (noisy) database in tabular form	+
$\mathcal{T}$	Exploration tree, ie a tree containing n-grams up to length n where the virtual root is at level 0, 1-grams at level 1, etc.	*
e	An event, ie an occurrence of an identifier at a location and time	+
$v \in \mathcal{T}$	A node in $\mathcal{T}$ That contains one or more events with the same key	*
$key(v)$	Returns the key, or unique identifier, associated with the node. In terms of ngram, this is the location. In terms of ngram2.0, this is a location/time pair	*+
$key(e)$	Returns the key associated with the event in the same manner as $key(v)$	+
$id(e)$	Returns the identifier associated with the event	+
$\mathcal{U}$	The universe of all $key(v)$ Results, ie every unique identifier	*+
$g(v)$	N-gram associated with the node v, ie the sequence of $v \in \mathcal{T}$ from the tree root to v	*
$\lvert g(v) \rvert$	Real n-gram count of $g(v)$	*
$c(v)$	Noisy count of $g(v)$, ie $c(v) = \lvert g(v) \rvert + Lap(\lambda)$	*+
$levelset(i)$	Returns the set of nodes at level i of $\mathcal{T}$	*
$level(v)$	Returns the level of the node $v \in \mathcal{T}$. Note the root is at level 0	*
$\Pr(v)$	The probability of transiting from v's parent to v, ie $\Pr(v) = c(v) / \sum_{v_s in\ levelset(level(v))} c(v_s)$. That is, the normalized count of v and its siblings	*+
&	A special location that is placed at the end of a sequence (or Event placed at the end of a trajectory)	*+

* Chen et al ngram
+ This paper, ngram2.0

Table 4.2 Terminology

An event is an occurrence of an identifier at a location and time. Thus it holds a timestamp, location, and identifier(s). Note again that this thesis is focused on trajectories and not sequences, as compared in Section 1.3. A tabular database is a multiset of events which can be formed into trajectories to create a trajectory database. A sequence database is composed of a multiset of sequences, which can easily be formed into trajectories. A node can be of three types: root node, normal node, or leaf node (note unless otherwise specified the word node is used to denote a node in general).

These terms will be used in the further sections for discussing the algorithm used. The first phase of the algorithm is creating the exploration tree from n-grams calculated from the raw dataset.

4.3 Preprocessing

Absent in ngram is a preprocessing phase so this is a new phase in ngram2.0. It is shown in Algorithm 4.1 and displays a difference from ngram in that this algorithm can take in either a trajectory or a tabular dataset, instead of just a tabular dataset. If there is a tabular dataset as input, it is first converted into trajectories as in lines 2-6. This simply makes one trajectory for every unique identifier pair. It could be that this combines independent trajectories from the same unique identifier pair, but adding logic to find out which trajectories are independent would be a research effort in itself since the data being used is often sparse at times. More on the data restrictions is discussed in Section 6.1. Otherwise, each trajectory is added to the output dataset.

In both cases, each timestamp is bucketized based on the input time sensitivity parameter. This parameter is in seconds, so values of $t = 1, 60, 900, 3600, 21600, 86400$ truncates the timestamps to the second, minute, 15 minute, hour, 6 hour, day, respectively. This is done to help ensure there is enough data quality, i.e., that there are high quality n-gram counts. With a larger t, less time information is retained, but there are also fewer unique events in the universe and thus counts that would have been distributed across different events can be combined. For example, if just spatial data is used, each spatial point is one location. If spatiotemporal data is used with $t = 60$, each of these locations is now a different event based on which time bucket it is in. If there were originally 2,000 locations and the data is sampled over 24 hours, there is now a maximum of 48,000 unique locations/events since each location is now actually up to 24 different events.

```
Input:    Raw sequential database $\mathcal{D}$ or $\mathcal{D}'$
Input:    Time sensitivity $t$
Output:   Processed raw database $\mathcal{D}_o'$
Output:   Universe $\mathcal{U}$
1:     if input is $\mathcal{D}$:
2:         foreach row in $\mathcal{D}$:
3:             Convert time based on $t$
4:             Add $e = event(row)$ to $\mathcal{D}'$ at $id(e)$
5:             if $key(e) \notin \mathcal{U}$:
6:                 Add $key(e)$ to $\mathcal{U}$
7:     else if input is $\mathcal{D}'$:
8:         foreach trajectory in $\mathcal{D}$:
9:             Convert time based on $t$
10:            Add trajectory to $\mathcal{D}'$
11:            for $e \in trajectory$, if $key(e) \notin \mathcal{U}$:
12:                Add $key(e)$ to $\mathcal{U}$
13:    Add & as last event in each trajectory t
14:    $\forall$ t $\in \mathcal{D}'$, add n − grams in $t_0 - t_{\min(|t|, lmax)}$ to $\mathcal{T}$
15:    return $\mathcal{D}', \mathcal{U}$,
```

Algorithm 4.1 ngram2.0 preprocessing

Continuing Example 4.1, the resulting trajectories from running the events in Figure 4.1 through Algorithm 4.1 are shown in Figure 4.1. The events from Table 4.1 have been bucketized with $t = 3600$, or an hour sensitivity. This reduces the 6 different timestamps to 2 unique timestamps, reducing the overall universe size from 11 to 5. For comparison, Figure 4.2 shows the original trajectories without any time bucketization. Note that L_1 is LIB-234BB1034M, L_2 is LIB-234CA1075S, L_3 is LIB-234CA1241A, t_2 is 2017-07-07T13:… times, and t_2 is 2017-07-07T14:… times.

ID	Trajectory
1	$L_1t_1 \rightarrow L_2t_1 \rightarrow L_3t_2$
2	$L_1t_1 \rightarrow L_2t_1$
3	$L_2t_1 \rightarrow L_1t_1$
4	$L_1t_1 \rightarrow L_2t_2 \rightarrow L_3t_2$
5	$L_2t_1 \rightarrow L_1t_1 \rightarrow L_3t_2$
6	$L_1t_1 \rightarrow L_2t_2 \rightarrow L_3t_2 \rightarrow L_1t_2 \rightarrow L_2t_2$
7	$L_2t_1 \rightarrow L_1t_2$
8	$L_2t_1 \rightarrow L_3t_2 \rightarrow L_1t_2 \rightarrow L_2t_2$

Figure 4.1 Example 1 processed raw trajectories with t=3600

ID	Trajectory
1	$L_1t_1 \rightarrow L_2t_2 \rightarrow L_3t_3$
2	$L_1t_1 \rightarrow L_2t_2$
3	$L_2t_1 \rightarrow L_1t_2$
4	$L_1t_1 \rightarrow L_2t_3 \rightarrow L_3t_4$
5	$L_2t_1 \rightarrow L_1t_2 \rightarrow L_3t_3$
6	$L_1t_2 \rightarrow L_2t_3 \rightarrow L_3t_4 \rightarrow L_1t_5 \rightarrow L_2t_6$
7	$L_2t_2 \rightarrow L_1t_4$
8	$L_2t_2 \rightarrow L_3t_3 \rightarrow L_1t_4 \rightarrow L_2t_5$

Figure 4.2 Example 1 processed raw trajectories without time sensitivity

Now there is a trajectory dataset that has been modified to time buckets. The last step of this preprocessing phase is to calculate n-gram counts from the resulting trajectories.

4.3.1 Computing n-gram Counts

Computing n-grams counts is one of the easier concepts, but is not described in the original paper though it is implemented in ngrampy. The first of three phases in ngrampy is to decompose the raw dataset into n-grams. First it reads in all the sequences, truncating them at $\ell_{\max}$. Then it adds the special terminating symbol, &, at the end of the sequence. Note that this could mean that the sequences are actually $\ell_{\max}+1$ in length, but since & is just shows the end of the sequence, this technique presumably follows the ngram $\ell_{\max}$ requirement.

Ngram2.0 follows ngrampy's method of computing all n-grams prior to their use so the n-gram retrieval in line 5 of **Error! Reference source not found.** is a simple lookup in a ap structure. The technique is slightly different, though. Instead of two passes on the trajectories, there is just one which iterates through each trajectory terminating to $\ell_{\max}$, adding the &, and counting n-grams in one pass. Thus being more efficient than ngrampy.

2-gram	Count
$L_1t_1 \rightarrow L_1t_1$	0
$L_1t_1 \rightarrow L_2t_1$	2
$L_1t_1 \rightarrow L_1t_2$	0
$L_1t_1 \rightarrow L_2t_2$	2
$L_1t_1 \rightarrow L_3t_2$	1
$L_1t_1 \rightarrow \&$	1

2-gram	Count
$L_2t_1 \rightarrow L_1t_1$	2
$L_2t_1 \rightarrow L_2t_1$	0
$L_2t_1 \rightarrow L_1t_2$	1
$L_2t_1 \rightarrow L_2t_2$	0
$L_2t_1 \rightarrow L_3t_2$	2
$L_2t_1 \rightarrow \&$	1

1-gram	Count
L_1t_1	6
L_2t_1	6
L_1t_2	3
L_2t_2	4
L_3t_2	5
&	0

Table 4.3 1-grams for Example 1

2-gram	Count
$L_1t_2 \rightarrow L_1t_1$	0
$L_1t_2 \rightarrow L_2t_1$	0
$L_1t_2 \rightarrow L_1t_2$	0
$L_1t_2 \rightarrow L_2t_2$	2
$L_1t_2 \rightarrow L_3t_2$	0
$L_1t_2 \rightarrow \&$	1

2-gram	Count
$L_2t_2 \rightarrow L_1t_1$	0
$L_2t_2 \rightarrow L_2t_1$	0
$L_2t_2 \rightarrow L_1t_2$	0
$L_2t_2 \rightarrow L_2t_2$	0
$L_2t_2 \rightarrow L_3t_2$	2
$L_2t_2 \rightarrow \&$	2

Table 4.4 2-grams for Example 1

Continuing with Example 4.1, part of the resulting n-gram counts can be seen in Table 4.3 and Table 4.4. These tables contain only 4 sets of the 5 2-grams and also do not display the 3-grams or the 4-grams. Taking $L_2t_2 \rightarrow L_3t_2$ as an example, this 2-gram occurs in Figure 4.1 containing the last 2 events of trajectory 4 and the 2nd and 3rd events of trajectory 6. Thus it has a count of 2.

This section discussed the computation of n-gram counts, which is a data preprocessing step before the algorithm in Algorithm 4.2 is underway. The first algorithmic step is to create the exploration tree by computing ε for distribution over the entire exploration tree to add noisy events to it.

4.4 Exploration Tree Creation

The first step of ngram2.0 is the exploration tree construction step. The main difference in ngram2.0 is that it uses two different state variables and rounds budget calculations to fix infinite budget calculation problems. This is the core of the algorithm, which is to create n-grams based on the input dataset then add noise to these n-grams. The ngram version is described first and is detailed in Algorithm 4.2, which is adapted from [51].

The exploration tree starts with one node in it, the root at level 0. Then each level is expanded until the tree is of height $n_{\max}$ +1. Note that in line 2, the $levelSet(i)$ is first called on level 0 which contains the root of the tree. Children are added to the root and expanded further thereafter.

More specifically, the algorithm in Algorithm 4.2 iterates from 0 to $n_{\max}$, expanding each tree node in the current level by adding children to each node dependent on a noisy count value. The first step of this algorithm, line 3, is to compute the privacy budget ε_{vij} value for each node expansion (Section 0). In lines 4-6 the noisy count for each possible child is computed (Sections 4.4.3). Each of these nodes are added as a child to the current node. Note that children are default added as nodes so that they can be further expanded. The final step, lines 7-10, is to check if each node has passed the threshold θ. If a node does not, it is marked as a leaf so it is not further expanded (Section 4.4.1). This check is based off the noisy counts calculated from the Laplace distribution in line 6.

```
Input:    Raw sequential database $\mathcal{D}$
Input:    Universe $\mathcal{U}$
Input:    Privacy budget $\varepsilon$
Input:    Maximal sequence length $\ell_{\max}$
Input:    Maximal n-gram length $n_{\max}$
Input:    Threshold $\theta$
Output:   Noisy exploration tree $\tilde{\mathcal{T}}$
1:    for $i < n_{\max}$;
2:        foreach non-leaf node $v_{ij} \in levelSet(i)$ && $v_{ij} != \&$:
3:            Compute $\varepsilon_{vij}$;
4:            $U_c \leftarrow$ all possible children with ids $\mathcal{U} \cup \{\&\}$;
5:            // Compute noisy count for each $u_k \in U_c$
6:            $Q = \{|g(u_1)|, |g(u_2)|, \ldots, |g(u_{|\mathcal{U}|-1})|\}$;
7:            $\tilde{Q} = \{c(u_1), c(u_2), \ldots, c(u_{|\mathcal{U}|-1})\}$;
8:            foreach $u_k \in U_c$:
9:                Add $u_k$ to $\mathcal{T}$;
10:               if $c(u_k) < \theta$:
11:                   Mark $u_k$ as leaf;
11:   return $\tilde{\mathcal{T}}$;
```

Algorithm 4.2 ngram exploration tree creation

Ngram2.0 follows the logic of ngram for this phase, but implements it slightly differently in order to make sure certain checks are followed, as shown in Algorithm 4.3. The algorithm still iterates from 0 to $n_{\max}$, expanding each tree node in the current level by adding children to each node dependent on a noisy count value.

```
Input:  Raw sequential database $\mathcal{D}'$
Input:  Universe $\mathcal{U}$
Input:  Privacy budget $\varepsilon$
Input:  Maximal sequence length $\ell_{\max}$
Input:  Maximal n-gram length $n_{\max}$
Input:  Threshold $\theta$
Output: Noisy exploration tree $\tilde{\mathcal{T}}$
1:   for $0 \leq i < n_{\max}$:
2:     foreach non-leaf node $v_{ij} \in levelSet(i)$ && $v_{ij} != \&$:
3:       if $type(v_{ij})\ is\ root$:
4:         Set node budget $\varepsilon_{ij}$ to $\varepsilon / n_{\max}$
5:       Compute $\theta$ for $v_{ij}$                                  > See 4.4.1
6:       Compute $\varepsilon_{vij}$                                       > See 4.4.2
7:       $U_c \leftarrow$ all possible child events with ids $\mathcal{U} \cup \{\&\}$
8:       //  Compute noisy count for each $u_k \in U_c$                 \
9:       $Q = \{|g(u_1)|, |g(u_2)|, \ldots, |g(u_{|\mathcal{U}|-1})|\}$                  > See 4.4.3
10:      $\tilde{Q} = \{c(u_1), c(u_2), \ldots, c(u_{|\mathcal{U}|-1})\}$                 /
11:      foreach $u_k \in U_c$:
12:        Add $u_k$ to $\mathcal{T}$
13:        Set $u_k$ leaf/released states w.r.t. $c(u_k)$, $\theta$, & $\varepsilon_{ij}$
14:      foreach $u_k \in U_c$:
15:        Set node budget $\varepsilon_{ij}$ adaptively                     > See 4.4.4
16:  return $\tilde{\mathcal{T}}$
```

Algorithm 4.3 ngram2.0 exploration tree creation

The loops in lines 1-2 iterate the same as ngram. Lines 3-4 shows the step of budget calculation for the first loop iteration. This is described in ngram but only in theory. The next step, line 5, is the threshold computation (Section 4.4.1). The only difference is that ngram2.0 has a slightly different check for if there is more budget left to expand more nodes. Then line 6 is the computation of the privacy budget ε_{vij} value for each node expansion (Section 4.4.2). The difference in ngram2.0 is that it uses a more programmable $P_{\max}$ calculation and as ngrampy does, uses a ceiling function to round decimals to integers for the ε_v calculation. Line 7 creates a new universe that also contains the termination event so that it can be considered when expanding nodes as ngram.

In lines 8-10 the noisy count for each possible child is computed (Sections 4.4.3) as ngram, though minor changes are discussed in Section 4.4.3. Each of these nodes are added as a child to the current node. Note that children are default added as nodes so that they can be further expanded. Note that in line 10, the child node u_k with event & is set to a count of 0 since it is not placed in the tree for its count but only for transitional information so that the current n-gram ends at that event and is not further expanded.

The nodes with noisy counts are then added to the exploration tree and then their states are set (Section 4.4.4), in lines 11-13. Line 13 is an additional step added in ngram2.0 and is based both on the noisy count and budget of the node. It also shows that there are actually two different state variables. Ngrampy ditched using the leaf states and opted for using a bit associated with each node as to whether it is released or not. Ngram2.0 decided to combine the two methods. The released flag is the main flag that is used and it is used in getting the Markov parent and enforcing consistency constrains. The type parameter is only used in the exploration tree creation phase as seen in line 2 of Algorithm 4.3. This conditional statement makes sure that if the nodes is marked as a leaf, it is not further expanded.

The final step, line 14-15, is to set the budget for all the nodes just added. This is a step in ngram, but the structure of having these two loops separate is a change in ngram2.0 and as shown ngram2.0 explicitly shows where the budget is set, unlike ngram. For most nodes the budget is set at the very end after adding it to the tree, lines 14-15. Then it is used in the next iteration, since the children of the current iteration become the nodes iterated over in the next iteration. As mentioned earlier, the only exception is for the root since that is expanded in the first iteration. Thus the budget must be assigned to it, as in lines 3-4. This loop in lines 14-15 is separate from the previous loop in lines 11-13 because of the Markov parent calculation when calculating the budger for 1-grams. In this case, the Markov parent is the root node so the children of the Markov parent are the 1-grams. Since the counts of these children are used, all of them need to be added to the tree before being used. Thus the loop is separate, even though this does not matter for n-grams greater than 1-grams.

Another difference that is not shown here is that ngram2.0 is geared to work with spatiotemporal events instead of just spatial points like ngram. This affects the data input and event comparison code, but will not necessarily show up as a difference in the algorithm listing in this Chapter.

Continuing with Example 4.1, Figure 4.3 shows what a resulting noisy exploration tree could look like after this phase has finished. Because of space constraints, the tree figures for this example do not look exactly like a tree and some nodes are vertical while others are shown horizontal. For example, all of nodes v_1 through v_6 are all on level 1. The figure also shows which nodes are marked as leaf nodes or released. Note that because of the logic taken by ngram2.0, node v_6 is not marked as released, but it could be as ngrampy does. This slight change does not affect the logic in future phases so it has not been changed.

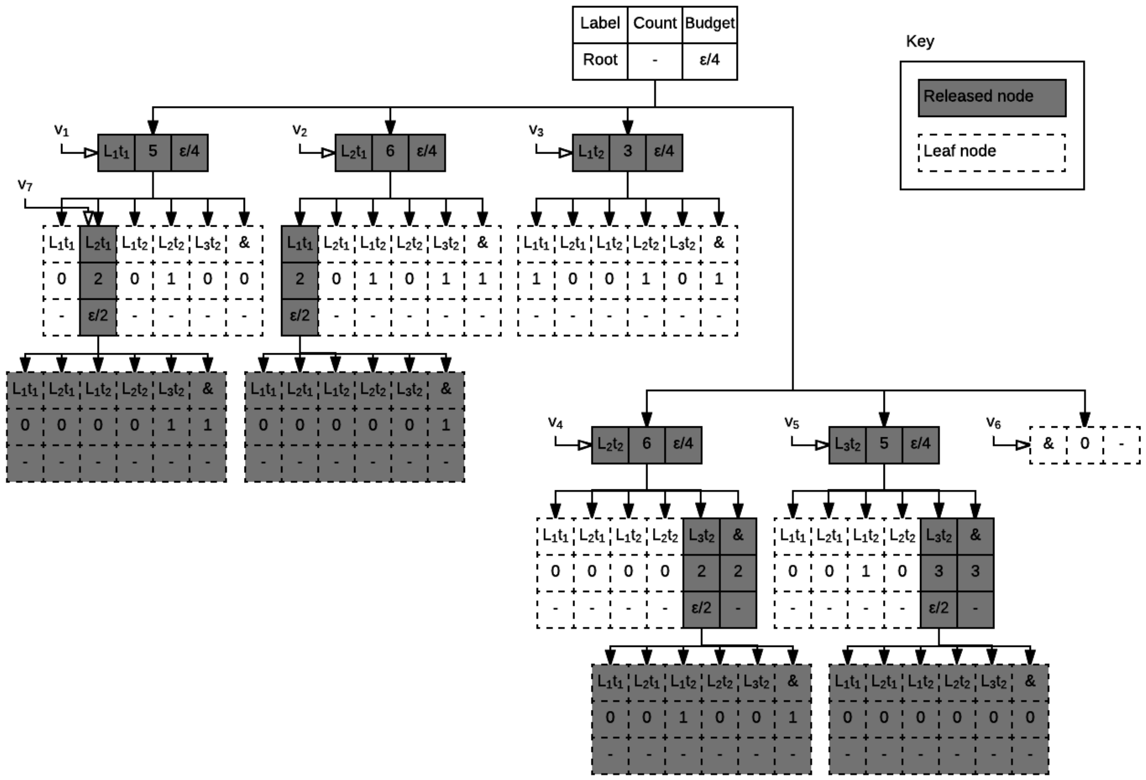


Figure 4.3 Example 1 noisy exploration tree

There are four steps from Algorithm 4.3 that will be described in the following sections. The first step is the computation of the threshold parameter for each node.

4.4.1 Computing θ

The threshold θ is essential for keeping the exploration tree from being expanded too much. If $c(v)$ of any node v is less than θ it is set to a leaf node so that it is not further expanded. Thus nodes with small counts do not introduce excessive n-gram noise in $\mathcal{T}$.

The ngram algorithm gives the following equation given the threshold θ and total privacy parameter ε [51]

$$P_\theta = \int_\theta^\infty \frac{\varepsilon}{2\ell_{max}} \exp\left(-\frac{x\varepsilon}{\ell_{max}}\right) \; dx = \frac{1}{2} \exp\left(-\frac{\varepsilon\theta}{\ell_{max}}\right)$$

which is the probability density function of the Laplace distribution, as described in Section 3.1.1. A node will generate $|\mathcal{U}|P_\theta$ false nodes, a node with real count 0 but noisy count greater than θ, at each expansion where P_θ is the probability of Laplace noise passing θ. Thus the above equation gives the probability that a random variable sampled from the Laplace distribution will be θ or greater. This then gives [51]

$$\theta = \ell_{max} * \ln\left(\frac{|\mathcal{U}|}{2}\right) / \varepsilon \tag{4.1}$$

which is straightforward to implement. Note that whether or not a node is marked as a leaf, it is added to $\mathcal{T}$. The leaf marking is checked in the next exploration tree creation iteration as to whether that nodes should be expanded.

Interestingly, ngrampy does not solely use θ to determine when to not expand a node. It also checks whether there are more levels left to expand based on the prediction in Section 4.4.2 when computing the adaptive budget.

Ngram2.0 does a similar check as ngrampy does to ensure that there still is budget left to expand nodes. It does not store the number of levels left, though. It stores the amount of budget left. So if this value is 0 or less, the node is marked as a leaf.

Continuing with Example 4.1, when expanding the root, the θ value is calculated as $\theta = \ell_{max} * \ln\left(\frac{|\mathcal{U}|}{2}\right)/\varepsilon = 6 * \ln\left(\frac{|6|}{2}\right)/5 = 1.318$. This is the same for node v_1. For expanding the 2-gram node v_7, the threshold is set to $\theta = 6 * \ln\left(\frac{|6|}{2}\right)/10 = 0.6592$, which uses the budget calculated for this node while expanding node v_1. This calculation proceeds for the other nodes in the same fashion.

How to compute the threshold was just described. The next section will describe how the budget for each node is computed.

4.4.2 Computing ε

The main parameter for dp is the privacy budget, or ε. This section will describe its computation. The input ε indicates the total budget that can be used by the algorithm, but this budget must be distributed across the algorithm as the synthetic data is being created. This calculation follows the sequential and parallel composition properties as laid out in Section 3.1.1. First, each level uses a portion of the remaining ε value since they are dependent sections of the tree. This follows the sequential composition property. Second, each node within a level uses the same ε value since each node is disjoint from the others. This follows the parallel composition property. The following will show specifically how the budget is divided.

Each n-gram in a level of the exploration tree could use the same privacy budget value ε /$n_{\max}$ if it is assumed that every n-gram extends to a tree height of $n_{\max}$. This provides an easy calculation, but would waste part of the budget, adding unnecessary noise since many branches do not extend to $n_{\max}$. Thus ngram uses an adaptive privacy budget allocation scheme which is a way to estimate the height of the subtree rooted at any node v so that a proper amount of budget is used per node in this subtree [51]. This adaptive model predicts the height of each subtree that is being expanded based off the probability, $P_{\max}$, of moving from the current node v to the child with the highest noisy count. This $P_{\max}$ value is estimated using the Markov assumption (see Section 3.2). So the sequence can be predicted by the last n events, i.e., the probability from location at position 1 to i can be estimated by a probability starting at location at position 2 until i: $P(L_{i+1} \middle| L_i^1) :\approx P(L_{i+1} \middle| L_i^2) :\approx P(L_{i+1} \middle| L_i^3) :\approx \cdots :\approx P(L_{i+1})$. The leftmost probability in this chain is desired since it contains the most information, but not all parts of the chain will be present in the tree since nodes are only expanded if they pass θ. Also note that there is always an approximation since all 1-grams are in the tree.

For example, if the probability of the sequence $L_1 \rightarrow L_4 \rightarrow L_2 \rightarrow L_1$ is desired, the probabilities of occurrence of $L_4 \rightarrow L_2 \rightarrow L_1$, $L_2 \rightarrow L_1$, or L_1 can be used, depending on which is present in the exploration tree.

For expanding level 0, $\varepsilon / n_{\max}$ has to be used since there is no information known about the counts of any n-grams. For levels $i \geq 1$, the adaptive budget allocation scheme is used. It starts with calculating $P_{\max}$, which is calculated as [51]

$$P_{\max} = \max_{L_{i+1} \in \mathcal{U} \cup \{\&\}} \check{P}(L_{i+1} \middle| L_i^1) \tag{4.2}$$

where $\check{P}$ is the leftmost probability mentioned above, or the estimation of $P(L_{i+1} \middle| L_i^1)$.

Based on the probability $P_{\max}$, noisy count of the current node $c(v)$, and threshold θ predicting the amount of levels that will be expanded, the privacy budget can be computed as [51]

$$\varepsilon_v = \frac{\bar{\varepsilon}}{\min\left(\log_{P_{\max}} \frac{\theta}{c(v)}, n_{max} - \mathrm{i}\right)} \tag{4.3}$$

where $\bar{\varepsilon}$ is the original privacy budget minus all budgets used by v and v's ancestors. This budget ε_v is used to add noise to v's children. Note every branch, root to leaf path, uses the full ε value. The proof for these formulas can be viewed in the original paper by Chen et al.

The code in ngrampy uses Equation (4.3 except for one difference. If $P_{\max} = 1$ then the denominator is the second term, $n_{max} - \mathrm{i}$. Otherwise the min function is used. This is presumably because $\log(1) = 0$, making the log value ∞. In a minor change, ngrampy labels the denominator as the number of levels left for expanding that node. This gives a clearer understanding about how that equation works. The resulting ε value is a fraction of the remaining ε values based on the number of levels left. Ngrampy implements Equation (4.2) straightforwardly.

In ngram2.0, this calculation formula is slightly expanded, still maintaining the calculation but in a programmable way. Equation (4.2 is then

$$P_{\max} = \frac{\max\left(c\left(levelset\left(lev(v)\right)\right)\right)}{\sum_{v_i \in levelset(lev(v))} c(v_i)} \tag{4.4}$$

which is the normalization of the maximum noisy count child value. For Equation (4.3, ngram2.0 does not check if $P_{\max} = 1$, because if this is the case then that term would be infinite and the , $n_{max} - \mathrm{I}$ term would be chosen anyway.

Equation (4.3) also is changed slightly to

$$\varepsilon_v = \frac{\bar{\varepsilon}}{\min\left(\mathrm{ceil}\left(\log_{P_{\max}} \frac{\theta}{c(v)}\right), n_{max} - \mathrm{i}\right)} \tag{4.5}$$

The ceiling function was added by ngrampy and is adopted in ngram2.0 so that excessive budget is not used. This is the case when the result of the first term of the min function is less than 1, which could result in a greater amount of budget than actually exists. It is unclear why this was not explicitly mentioned in ngram.

Continuing with Example 4.1, during the first iteration of Algorithm 4.3 the root node is being expanded and thus the budget is $\varepsilon/n_{max} = \varepsilon/4 = 5$. This is also true for the next iterations which are over the 1-grams. For expanding node v_7, the probability will be

$$P_{\max} = \frac{\max(6+6+3+4+5)}{6+6+3+4+5} = \frac{6}{24} = 0.25$$

which means that the budget will be

$$\varepsilon_v = \frac{\bar{\varepsilon}}{\min\left(\mathrm{ceil}\left(\log_{P_{\max}} \frac{\theta}{c(v)}\right), n_{max} - \mathrm{i}\right)} = \frac{10}{\min\left(ceil\left(\log_{0.25} \frac{1.31833}{2}\right), 2\right)}$$

$$= \frac{10}{\min(\mathrm{ceil}(0.30),2)} = 10$$

This process proceeds with the other nodes in a similar fashion.

Once the amount of budget for expanding the current node is computed, this node can be expanded. This is done by adding Laplacian noise to real n-gram counts.

4.4.3 Adding Laplacian Noise

The noisy counts are calculated by adding noise taken from the Laplace distribution to the raw n-gram counts calculated in Section 4.3.1. The logic behind using the Laplace distribution was discussed in Section 3.1.1. Note that the noise added can be positive or negative; the point is that the resulting synthetic numbers are centered around the original n-gram count distribution.

All items in the location universe are considered as children of the current node. Thus, if the current level is i, noise is added to all $(i + 1)$-grams with a base sequence from the root to the current node, the parent node, and all items from the universe as an ending event. Then noise is added to these n-grams. It is possible that some of the real counts of these n-grams are 0, so all events in the universe have a chance of being placed in the tree if the noise added brings the count above the θ threshold.

Ngram2.0 makes no major changes to this calculation as it is simply done. Yet, the implementation is a bit tricky, as will be discussed in the next Chapter, specifically Section 5.3.3.

Continuing with Example 4.1, the original n-grams can be seen in Table 4.3 and Table 4.4 (Section 4.3.1), while the noisy grams are in Figure 4.3 (Section 4.4). Since this is a simple addition of n-gram values to floating point values, there is nothing further that needs to be explained here.

Adding Laplacian noise creates new n-grams, but keeping and expanding all these new nodes would lead to very poor utility. Thus, a threshold is used to decide whether new nodes are expanded or not, which was described in Section 4.4.1. After this is all done, the last step is setting the two node states.

4.4.4 Setting Node States

When a node is added to the exploration tree, two states are set for it, as shown in Algorithm 4.4. This is based on the threshold and budget values. A node is marked as a leaf if its noisy count is less than the threshold or the budget of its parent node is 0. Then this node will no longer be expanded. This is used for the fill exploration tree phase to check if the node is the root node and to decide whether to expand a node. A node is marked as released if its noisy count is greater than the threshold or the budget of its parent node is 0. This is used for the Markov parent calculation and the enforce consistency constraints phase.

```
1:    if $c(u_k) < \theta$ || budget of $v_{ij} \leq 0$:
2:        Mark $u_k$ as leaf
3:    if $c(u_k) > \theta$ || budget of $v_{ij} \leq 0$:
4:        Mark $u_k$ as released
```

Algorithm 4.4 Setting the state of nodes

This phase has created a full synthetic exploration tree based on noisy n-gram counts drawn from the original raw counts. In order to improve utility, the next phase ensures that this resulting exploration tree looks like a natural tree created from raw data.

4.5 Enforce Consistency Constraints

It is presumed that the resulting noisy exploration tree does not follow the characteristics of a real exploration tree since noise was added irrespective of such real-life characteristics. Namely, the sum of all children's counts should not exceed that of their parents and leaf nodes noisy counts could be less than θ and thus be missing. Therefore, consistency must be enforced. The original ngram paper has more formal algorithmic equations [51], yet it was not totally clear what steps would have to be taken, so the following aims to clearly explain how the enforce consistency constraints step in ngram functions. This phase's main features are detailed in Algorithm 4.5.

```
Input:   Raw sequential database $\mathcal{D}$
Input:   Privacy budget $\varepsilon$
Input:   Maximal sequence length $\ell_{\max}$
Input:   Maximal n-gram length $n_{\max}$
Input:   Threshold $\theta$
Output:  Consistent exploration tree $\tilde{\mathcal{T}}'$
1:   for $v \in$ levelset(i), $\forall\ 1 \le i < n_{max}$:
2:       V+ = $\{children(v) > \theta\}$
3:       V-  = $\{children(v) < \theta\}$
4:       if V+ is empty:
5:           $c(v) = 0$;
6:       if V- is empty:
7:           $c(v) = \Pr(v) * c(v's\ parent)$;
8:       else:
9:           Compute $A(v_j)$ and $A(v_j)$;
10:          Renormalize c(v);
11:  return $\tilde{\mathcal{T}}'$;
```

Algorithm 4.5 ngram enforce consistency constraints code listing

This algorithm loops over all n-gram lengths in the tree, $1 \leq \text{i} < n_{max}$, or every node in every level except the last level since the nodes in the last level have no children. For every node $v \in levelset(i)$, get all children that pass the threshold theta, V^{+}, and all children that do not, V^{-}, lines 2-3.

If none of the children pass θ, lines 4-5, all children are assigned a noisy count of 0 since this is evidence v should not be expanded.

If all of the children pass θ, lines 6-7, it calculates the conditional probability of the existence of v and multiplies that by the noisy count of v's parent. That is $\Pr(v) * c(v'sparent)$. This step is actually the same as the renormalization step in line 10, but is for some reason described separately by ngram. Renormalization is described below.

Otherwise, in lines 8-10 there is a mix of passing and not passing children. It approximates the noisy counts of these children with the Markov assumption, i.e., that the future state only depends on the current state [51], and then normalizes these counts.

In line 9, the Markov noisy counts before normalization are denoted $A(v)$. The next few paragraphs describe the substeps and checks required for this step. Note that the notation v_c denotes a child of the current node v, v_i is a child in V^{-}, and v_j is a child in V^{+}.

Within this last step, if the level of the Markov parent is ≥ 2, meaning not a 1-gram, then the Markov parent is of enough quality to be used, so compute the following [51]:

$$A(v_i) = r_{v_i} * \sum_{v_j \in V^+} A(v_j) \tag{4.6}$$

That is, compute $A(\mathrm{v_i})$ as a ratio times the sum of all $A(v_j \in V^+)$ where in this case $\mathrm{A(v_j)}$ is just the noisy counts of the $\mathrm{v_j}$ nodes, or the nodes in the current level that pass θ. The ratio, r_{v_i}, in this case is [51]:

$$r_{v_i} = \frac{P(C(v_i))}{\sum_{v_j \in V^+} P\left(C(v_j)\right)} \tag{4.7}$$

where $C(v_i)$ is the Markov parent (not to be confused with the count variable $c(v_i)$). This ratio is the normalized count of all Markov parents of $v_i \in V^-$. The Markov parent is the (n-1)-gram, or largest n-gram if the (n-1)-gram does not exist, that could be used to predict the current node. For example, if the current node's key is $L_2t_1 \rightarrow L_1t_3 \rightarrow L_2t_4$ then the ideal Markov parent would be the (n-1)-gram $L_1t_3 \rightarrow L_2t_4$.

Otherwise, the Markov parent is a 1-gram so compute a value $A(\mathrm{v_i})$. If the sum of all $A(v_j \in V^+) \leq c(v)$ then compute all $A(\mathrm{v_i})$ as [51]:

$$A(v_i) = \frac{c(v) - \sum_{\mathrm{v_j} \in V^+} \mathrm{A(v_j)}}{|V^-|} \tag{4.8}$$

This makes sure that the sum of the noisy counts of all child nodes of v equals the parent noisy count, or $c(v)$. It does this by reserving the $c(v_j)$ counts (the $\sum_{\mathrm{v_j} \in V^+} \mathrm{A(v_j)}$ term) and evenly distributing the remaining $c(v)$ count value among all V^- children.

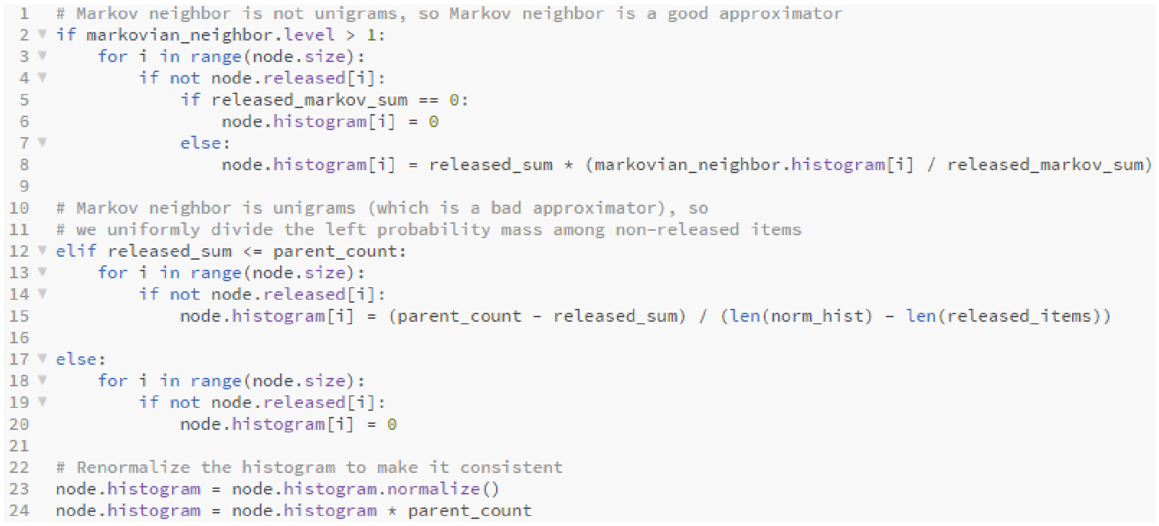
If none of these three steps hold (lines 4-5, 6-7, and the beginning of 8-9), set $\mathrm{A(v_j)} = 0$ because there are too many children counts. It is unclear why this step is included in ngram since it is taken care of in the inner workings of previous steps.

The final part, line 10, of this last else clause is where the Markov counts for each child are renormalized, as the following [51]

$$\forall v_k \in V, c(v_k) = c(v) * \frac{\mathrm{A(v_k)}}{\sum_{\mathrm{v_m} \in V} \mathrm{A(v_m)}}$$

Note that this is a redundant if the Markov parent is a 1-gram, as described above for lines 6-7. It is done every time nonetheless.

Note that ngrampy implements this enforce consistency constraints phase in the fill exploration tree phase, which would add some efficiency since it affects the number of nodes that are expanded. It implements it in a logically different way. The Python code is shown in Figure 4.4 (note this is not ngram2.0 code) where the histogram variable stores the noisy counts for the nodes. It also does the last renormalization step in every run no matter which conditional clause is taken, which simply reduces code complexity.



```python
# Markov neighbor is not unigrams, so Markov neighbor is a good approximator
if markovian_neighbor.level > 1:
    for i in range(node.size):
        if not node.released[i]:
            if released_markov_sum == 0:
                node.histogram[i] = 0
            else:
                node.histogram[i] = released_sum * (markovian_neighbor.histogram[i] / released_markov_sum)

# Markov neighbor is unigrams (which is a bad approximator), so
# we uniformly divide the left probability mass among non-released items
elif released_sum <= parent_count:
    for i in range(node.size):
        if not node.released[i]:
            node.histogram[i] = (parent_count - released_sum) / (len(norm_hist) - len(released_items))

else:
    for i in range(node.size):
        if not node.released[i]:
            node.histogram[i] = 0

# Renormalize the histogram to make it consistent
node.histogram = node.histogram.normalize()
node.histogram = node.histogram * parent_count
```

Figure 4.4 ngrampy enforce consistency constraints

The ngrampy approach is more programmatically logical, but ngram2.0 takes an approach closer to the ngram equations, as seen in Algorithm 4.6. Another change is that ngram2.0 uses the released state instead of the physically checking whether a node passed the threshold, as ngrampy does. This is just a quicker technique. As ngram, it iterates over all n-gram lengths in the tree, $1 \leq \text{i} < n_{max}$ and separates all children into V^+ and V^- of each $v \in levelset(i)$, lines 1-3.

The first check, lines 4-5, is the same, in that if there are no children that pass the threshold, clear all child counts. The logic of the rest is slightly different than ngram.

```
Input:    Raw sequential database $\mathcal{D}$
Input:    Privacy budget $\varepsilon$
Input:    Maximal sequence length $\ell_{\max}$
Input:    Maximal sequence length $n_{\max}$
Input:    Threshold $\theta$
Output:   Consistent exploration tree $\tilde{\mathcal{T}}'$
1:    for $v \in$ levelset(i), $\forall\, 1 \leq i < n_{max}$:
2:        $V^{+} = \{children(v) > \theta\}$
3:        $V^{-} = \{children(v) < \theta\}$
4:        if V+ is empty:
5:            $c(v) = 0$;
6:        else:
7:            if $level(Markov\ parent\ of\ v) \geq 1$ && $level(v) > 1$:
8:                Compute $r_v$ and $A(v)$ values for $V^{-}$
9:            else:
10:               if $\sum_{w \in V^{+}} c(w) \leq c(v)$:
11:                   Compute $A(v)$ values for $V^{-}$
12:               else:
13:                   Set $A(v) = 0$ values for $V^{-}$
14:           Renormalize c(v);
14:   return $\tilde{\mathcal{T}}'$;
```

Algorithm 4.6 ngram2.0 enforce consistency constraints

The second two steps in Algorithm 4.5, lines 6-10, are broken up and reduced into a more programmable structure in lines 6-14 of Algorithm 4.6. It does not check if all children do not pass the threshold (Algorithm 4.5 line 6) since regardless if all the children do not or none of them do not, the remaining noisy count left over from $c(v) - \sum_{v_j \in V^{+}} A(v_j)$ is distributed over the V^{-} children in the renormalization step, line 14. Also note that the check in line 7 means that any node v in levels less than 2 will not go through this consistency step since both nodes in level 1 and 2 return the root as the Markov parent, which is at level 0. The process of getting the Markov parent is described in detail in Section 5.4. Also note that if the current node v has no children, this whole phase has no effect.

If the Markov parent is not a 1-gram, compute $A(v_i)$ and r_{v_i} as before from Equation (4.6) and Equation (4.7). Otherwise, the Markov parent is a 1-gram, so as before calculate $A(v)$ or set all $A(v) = 0$ based on the sum of the passing nodes' counts.

The redundant renormalization check is removed, but instead renormalization is always done if the first check is not true. This reduces some complexity.

Continuing with Example 4.1, Figure 4.5 shows the exploration tree from Figure 4.3 after it has gone through this consistency phase. For the nodes in level 1 of the exploration tree in Figure 4.3, all of them are marked as released and the Markov parent's level is not ≥ 1 so the logic for each node falls into lines 10-13 of Algorithm 4.6. Node v_1 will be examined. First, the logical flow goes to lines 10-13 as stated. The sum of all released nodes is less than the parent count, $2 \leq 5$ since only one child (node v_7) is released and its count is 2, so line 11 is computed. All non-released children $A(v)$ counts are set to $(5 - 2) / 5 = 0.6$. The renormalization step sets the non-released counts to $5 * 0.6/5 = 0.6$ and one released child to $5 * 2/5 = 2$. Nodes v_2 and v_4 proceed similarly since for both the released child node count is less than the parent count. Node v_3 has no released children so line 5 sets all child counts to 0. Node v_5 has 2 released child nodes and their count is greater than the parent count, or $6 \leq 5$ is not true. So all non-released nodes are set to 0 and in the renormalization step, non-released nodes are set to $5 * 0/6 = 0$ while the 2 released nodes are set to $5 * 3/6 = 2.5$ since they have the same count. Node v_6 has no children since it is the termination sequence so this phase has no effect for that node.

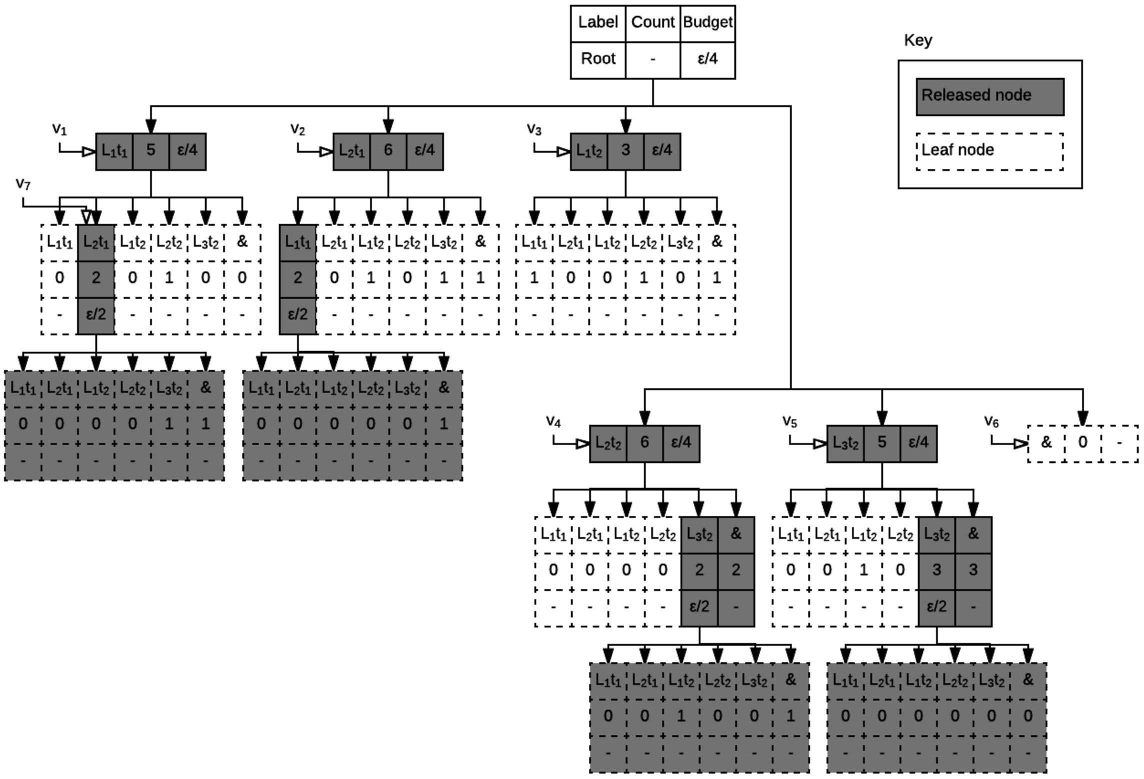


Figure 4.5 Example 4.1 consistent noisy exploration tree

Only 4 nodes in level 2 have children so only those nodes are affected in this phase. For example, node v_7 has a Markov parent of node v_2 which is in level 1 so line 8 calculations are done. There are no non-released child nodes, but the calculations for line 8 will be described. First, a Markov sum value is calculated as $2 + 0.8 + 0.8 + 0.8 + 0.8 + 0.8 = 6$. Second, the r_v value is calculated as $2\ /\ 6\ =\ 0.333333$ for the first child node and $0.8\ /\ 6\ =\ 0.133333$ for the other 5 child nodes. If there were non-released nodes, the $A(v)$ associated with them would be calculated as $A(v_i) = r_{v_i}\ *$

$\sum_{v_j \in V^+} A(v_j) = r_{v_i} * 2$ with the r_v values mentioned. None of that actually affects the released nodes, though. In the renormalization, 4 of the child nodes are set to $2 * 0/2 = 0$ while the remaining 2 nodes are set to $2 * 1/2 = 1$, which is all based off their noisy counts (0 for the first 4, and 1 for the remaining 2).

This phase has transformed the synthetic exploration tree into a structure that could resemble one created from raw data. The next and final stage transforms this exploration tree into a database of sequences.

4.6 Construct Synthetic Database

He previous phase has created a noisy n-gram exploration tree that resembles the qualities an exploration tree would have if made from real data. The last phase is to transform the n-grams from this tree into sequences, which is shown in Algorithm 4.7. The algorithm ngram2.0 adds in basic data processing steps and outputs both a sequential and a tabular dataset, instead of just a tabular one. As for the other phases, the ngram methodology will first be described and then the actual ngram2.0 algorithm.

For ngram in Algorithm 4.7, the first step in creating a synthetic database is to extend the current exploration tree so that sequences up to $\ell_{\max}$ can be generated. Otherwise, released sequences would have a maximum length of $n_{\max}$, which would not provide as much sequential data. This phase iterates through all of the longest n-grams currently in $\tilde{\mathcal{T}}$, joining all joinable paths to produce (n+1)-grams. Joinable n-grams g_1, g_2 are those where the nodes g_{12},g_{13}...g_{1n} match the nodes g_{21},g_{22}...$g_{2(n-1)}$ and n=|g_1|=|g_2| [51]. The resulting (n+1)-grams g_{11},g_{12}...g_{1n},g_{2n} are added to $\tilde{\mathcal{T}}$.

```
Input:   Consistent noisy exploration tree $\tilde{\mathcal{T}}'$
Input:   Privacy budget ε
Input:   Maximal sequence length $\ell_{\max}$
Input:   Maximal sequence length $n_{\max}$
Output:  Private sequential databases $\tilde{D}'$
1:   // First extend $\mathcal{T}$
2:   nodes = $levelSet(height(\tilde{\mathcal{T}}'))$;
3:   while nodes is not empty:
4:       foreach $v$ in nodes:
5:           if $v$ ☒ $v_i, \forall v_i \in nodes$ v:
6:               Extend $v$ with $v_i$;
7:           nodes = all nodes added;
8:   // Then generate synthetic database
9:   foreach $v \in levelset(i), \forall height(\tilde{\mathcal{T}}') > i > 0$:
10:      Add $c(v)$ occurrences of $g(v)$ to $\tilde{D}'$;
11:      Subtract all n-gram counts of g(v) in $\tilde{\mathcal{T}}'$;
12:  return $\tilde{D}'$;
```

Algorithm 4.7 ngram construct synthetic database code listing

This extended exploration tree can then be converted into a synthetic dataset of sequences. Starting from the new height of the tree, for every node $v \in levelset(i)$, add $c(v)$ occurrences of $g(v)$ to the output. Then subtract every n-gram count that supports $g(v)$, that is all n-grams with n from 0 to $level(v)$ that occur in $g(v)$ [51]. For example, if the following is the resulting sequence to be published $L_1 \rightarrow L_4 \rightarrow L_2$, then the noisy counts of the following 6 n-grams are decremented by 1: $L_1 \rightarrow L_4 \rightarrow L_2$, $L_1 \rightarrow L_4$, $L_4 \rightarrow L_2$, L_1, L_4, and L_2. Now ngram is finished and there is a synthetic sequential database.

Before completing the steps described for this phase, ngrampy has clean and sort steps. The clean step deletes all terminating sequence events and n-grams with $c(v) \leq 0$. The sort step reverse sorts the n-grams based off their lengths, which is presumably done to speed up the extension step so that the longest sequences can be found easily. It also adds a floor function in the extension step which floors all noisy counts, which up until this point were floating point numbers, to integers. It also does not add a sequence if subtracting the supporting n-gram counts would results in negative counts.

ngram2.0 follows the ngram algorithm with the ngrampy additions, the resulting algorithm being shown in Algorithm 4.8. Since these additions are practical and do not affect the algorithmic integrity, they will be discussed further in the implementation section. The one difference is that as the trajectory is added to the trajectory database $\widetilde{\mathrm{D}}'$, ngram2.0 assigs a unique identifier $\mathrm{hash}(\mathrm{key}(\mathrm{v}), \mathrm{seed})$ where $seed$ is a random number. The significance of this difference is that there is an explicit identifier assigned to each trajectory. The complexity of handling identifiers in this situation is discussed in Section 4.1. Once this is done, there is a database of sequences, $\widetilde{\mathrm{D}}'$, which is not the same format as an original tabular database so ngram2.0 also adds another step which is to generate the tabular database $\widetilde{\mathrm{D}}$, which is a simple iteration. For every event in every trajectory in $\widetilde{\mathrm{D}}'$, add that event as a row to $\widetilde{\mathrm{D}}$. Now the algorithm is finished and there is a synthetic tabular database. Note that with time bucketization, it may provide more utility using the trajectory database output since this orders the events in the trajectory precisely. With the tabular dataset, it can be unclear which event happens before or after another event for a particular ID since the time is generalized.

```
Input:   Consistent noisy exploration tree $\tilde{\mathcal{T}}'$
Input:   Privacy budget ε
Input:   Maximal sequence length $\ell_{\max}$
Input:   Maximal sequence length $n_{\max}$
Output:  Private sequential databases $\widetilde{D}'$, $\widetilde{D}$
1:   $\forall\, v \in \mathcal{T}$, floor(c(v))
2:   // First extend $\mathcal{T}$
3:   nodes = $levelSet(height(\tilde{\mathcal{T}}'))$;
4:   while nodes is not empty:
5:       foreach $v$ in nodes:
6:           if $v \boxtimes v_i, \forall v_i \in nodes$ v:
7:               Extend $v$ with $v_i$;
8:           nodes = all nodes added;
9:   // Then generate synthetic database
10:  foreach $v \in levelset(i), \forall\, height(\tilde{\mathcal{T}}') > i > 0$:
12:      Add $c(v)$ occurrences of $g(v)$ to $\widetilde{D}'$;
13:      Subtract all n-gram counts of g(v) in $\tilde{\mathcal{T}}'$;
14:  Create $\widetilde{D}$ from $\widetilde{D}'$;
15:  return $\widetilde{D}'$, $\widetilde{D}$;
```

Algorithm 4.8 ngram2.0 construct synthetic database code listing

Continuing with Example 4.1, Figure 4.6 shows the floored and extended version of the exploration tree from Figure 4.5. Note that in this case none of the sequences were actually extended. If, for example, the child of node v_2 with noisy count 2 was $L_3 t_2$, then the n-gram sequence starting at node v_1 not ending in & (ie $L_1 t_1 \rightarrow L_2 t_1 \rightarrow L_3 t_2$) could be extended with the sequence starting at node v_2 (ie $L_2 t_1 \rightarrow L_3 t_2 \rightarrow$ &).

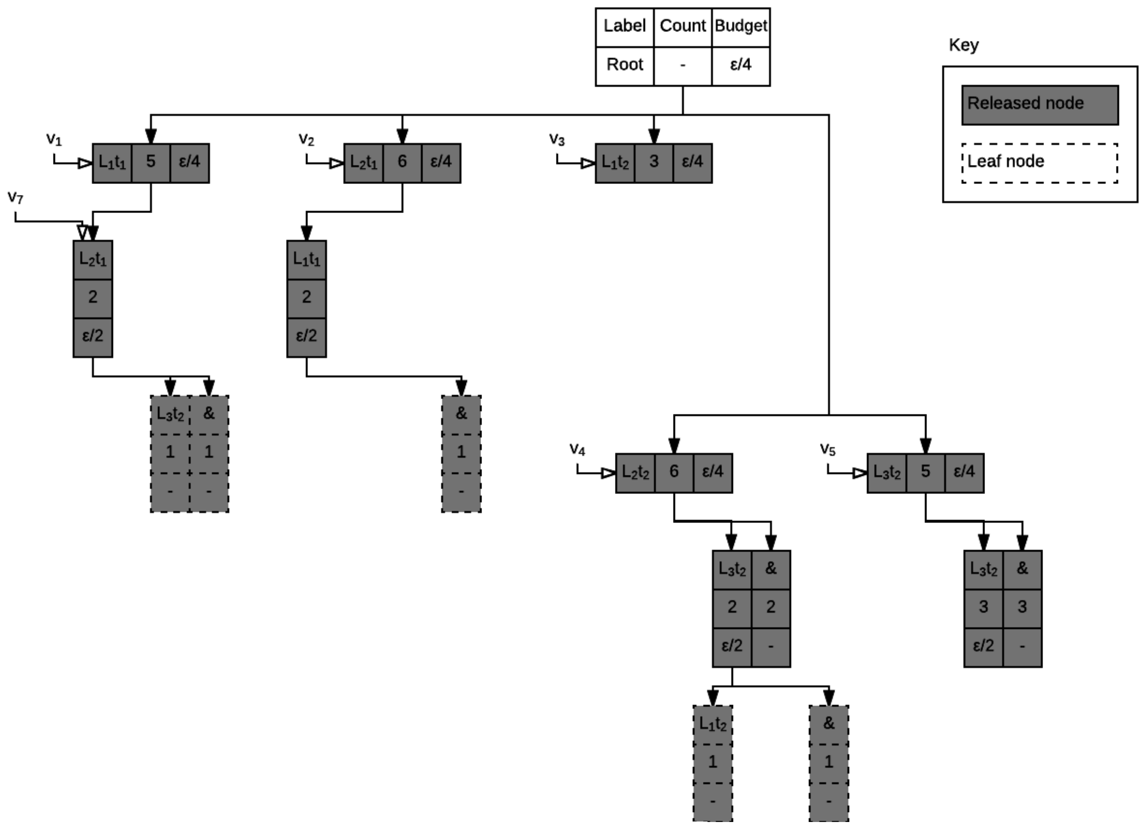


Figure 4.6 Example 4.1 extended noisy exploration tree

Now that the final version of the exploration tree is completed, the synthetic database can be created. Figure 4.7 shows the resulting synthetic trajectory database resulting from Figure 4.6.

ID	Trajectory	ID	Trajectory
1	$L_1 t_1 \rightarrow L_2 t_1$	11	$L_3 t_2 \rightarrow L_3 t_2$
2	$L_1 t_1 \rightarrow L_2 t_1 \rightarrow L_3 t_2$	12	$L_1 t_1$
3	$L_2 t_1 \rightarrow L_1 t_1$	13	$L_2 t_1$
4	$L_2 t_1 \rightarrow L_1 t_1$	14	$L_2 t_1$
5	$L_2 t_2 \rightarrow L_3 t_2$	15	$L_1 t_2$
6	$L_2 t_2 \rightarrow L_3 t_2 \rightarrow L_1 t_2$	16	$L_1 t_2$
7	$L_2 t_2$	17	$L_2 t_2$
8	$L_2 t_2$	18	$L_2 t_2$
9	$L_3 t_2$		
10	$L_3 t_2 \rightarrow L_3 t_2$		

Figure 4.7 Example 1 resulting trajectories

Finally, Table 4.5 shows the final synthetic dataset created by ngram2.0, with regards to Example 1. The ID hashes are elided. This would be in CSV format, but is shown in a table for simplicity.

Timestamp	AP Name	ID
2017-07-07T14:00:00Z	LIB-234CA1241A	5c42a3e38223c7c13ad39e95e489…
2017-07-07T14:00:00Z	LIB-234CA1241A	5c42a3e38223c7c13ad39e95e489…
2017-07-07T13:00:00Z	LIB-234BB1034M	1c7816ca342d04648cb4fabfe015…
2017-07-07T13:00:00Z	LIB-234BB1034M	1c7816ca342d04648cb4fabfe015…
2017-07-07T13:00:00Z	LIB-234BB1034M	165b5352f9c803326caeaeb23cdf…
2017-07-07T14:00:00Z	LIB-234BB1034M	165b5352f9c803326caeaeb23cdf…
2017-07-07T14:00:00Z	LIB-234BB1034M	1edac6a979acef029560d5af1c28…
2017-07-07T13:00:00Z	LIB-234CA1075S	4b1d3b759ec5e8949617dfb18257…
2017-07-07T13:00:00Z	LIB-234CA1075S	ae028b1755fda5bf37fd5ae7d2ce…
2017-07-07T13:00:00Z	LIB-234CA1075S	633fa612dddb53172d4ace92ac4c…
2017-07-07T13:00:00Z	LIB-234CA1075S	c164af584a4f034f58c657e513b8…
2017-07-07T14:00:00Z	LIB-234CA1075S	b7d17374c1e4682fe04d83313748…
2017-07-07T14:00:00Z	LIB-234CA1075S	e37ca4a5d3790ec34f710e721e26…
2017-07-07T14:00:00Z	LIB-234CA1075S	f4b416d9eeddcb9c2f024ffda1c8…
2017-07-07T14:00:00Z	LIB-234CA1075S	f4b416d9eeddcb9c2f024ffda1c8…
2017-07-07T14:00:00Z	LIB-234CA1075S	f4b416d9eeddcb9c2f024ffda1c8…
2017-07-07T14:00:00Z	LIB-234CA1075S	22f138742eb5261dff3359c43aea…
2017-07-07T14:00:00Z	LIB-234CA1241A	22f138742eb5261dff3359c43aea…
2017-07-07T14:00:00Z	LIB-234CA1241A	bbe86da9dd4eb04e90d0c86c5b01…
2017-07-07T14:00:00Z	LIB-234CA1241A	bbe86da9dd4eb04e90d0c86c5b01…
2017-07-07T13:00:00Z	LIB-234CA1075S	0edccb8b9643229ac7754118f948…
2017-07-07T13:00:00Z	LIB-234CA1075S	0edccb8b9643229ac7754118f948…
2017-07-07T13:00:00Z	LIB-234BB1034M	e4dc9e8b88c9b4a261b379e89b91…
2017-07-07T13:00:00Z	LIB-234BB1034M	e4dc9e8b88c9b4a261b379e89b91…
2017-07-07T14:00:00Z	LIB-234CA1241A	e4dc9e8b88c9b4a261b379e89b91…
2017-07-07T14:00:00Z	LIB-234CA1241A	e652ed30adcb431d975c6281391c…
2017-07-07T14:00:00Z	LIB-234CA1241A	cf2e8fffabbfc5f3b37cb9eba686…
2017-07-07T14:00:00Z	LIB-234BB1034M	d34c3aaf6052d1ca27d90ee21f0b

Table 4.5 Example 1 synthetic output authmgr tabular data

This last phase has converted the exploration tree into a database of sequences for ease of use by the resulting data mining entity. Thus the ngram2.0 algorithm is now complete. A raw dataset can now be transformed into a synthetic dataset that protects user privacy based on the formal guarantees of dp.

4.7 Conclusions

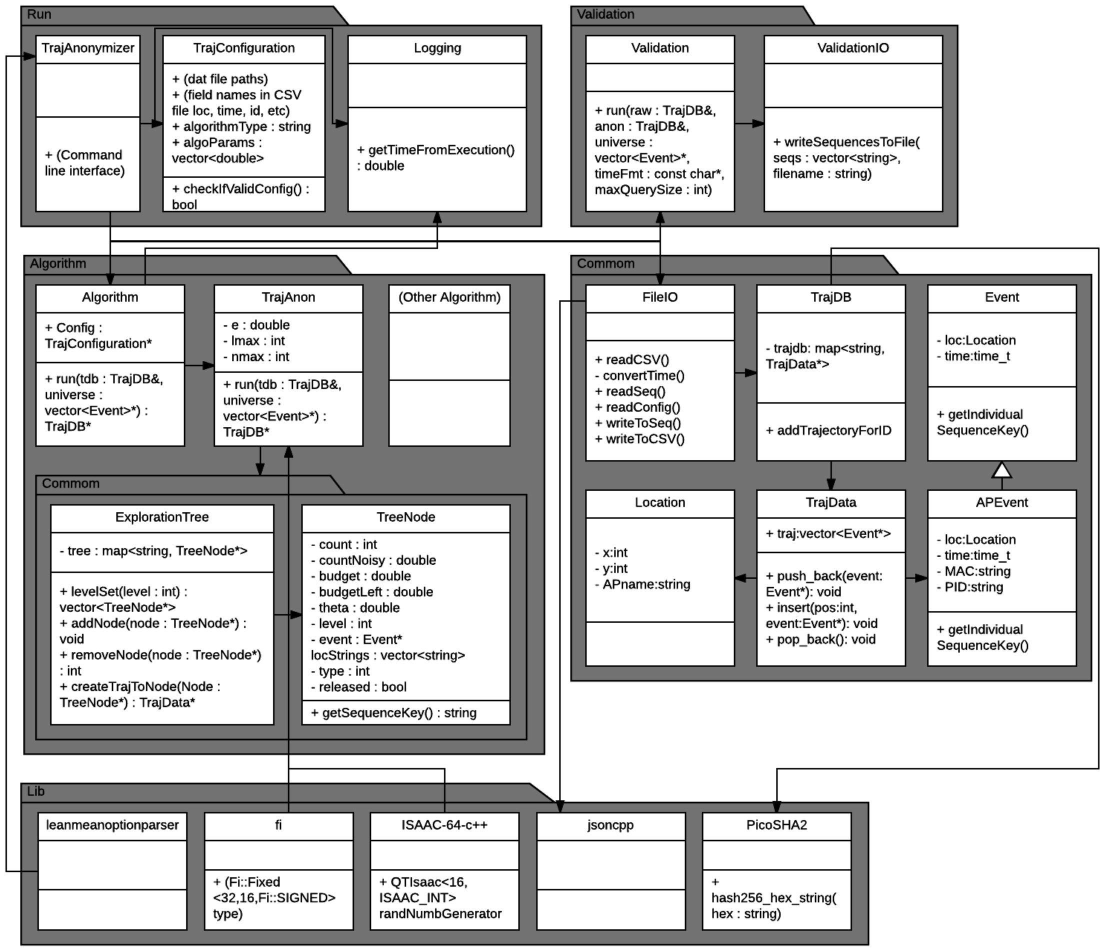
This chapter described the ngram2.0 algorithm. It did this by detailing the main phases of the ngram algorithm, along with relevant changes made by ngram2.0, both minor step changes and major phase changes. The main difference is in the data used. Ngram works with spatial data whereas ngram2.0 deals with spatiotemporal data. This is discussed more in the implementation Chapter. The algorithm starts by deconstructing a tabular dataset into n-grams. It then creates an exploration tree from noisy n-grams counts based on the Laplace distribution. This tree is then edited to conform to consistency constrains. Finally, the tree is deconstructed into trajectories and then a tabular database. The next chapter will go more in depth into the actual implementation strategy and details of ngram2.0.

5 Implementation

This chapter describes the implementation of the algorithm described in the previous chapter. First the code framework will be discussed inside of which the new ngram2.0 algorithm is placed. This framework is meant to provide a plug and play interface for PPTDP algorithms where the basic classes are already built, thus providing faster implementation. Second the algorithm implemented in this framework is discussed. As mentioned, this is named ngram2.0 as it is a slight upgrade from the ngram algorithm.

This chapter provides the details of the implementation that could later be analyzed and improved upon. The three main contributions of this chapter are as follows. First, the implementation of the handling of spatiotemporal data is discussed. Second, it shows the usage of fixed point numbers in the Laplacian calculations when adding noise as to not fall prey to floating point issues that leak information. Third, a method for finding the Markov parent is described.

A diagram based on UML (Unified Modeling Language) of the C++ classes created is given in Figure 5.1. This contains 5 packages (they are not actual packages in C++ but rather logical groupings of the classes). The following will be used to refer to classes: {package name(s)}/{class name}. The Run package handles running all other classes. The main entry point to the code is in the Run/TrajAnonymiser class. This contains the command line interface (CLI). It calls classes in the Algorithm and Validation packages. The former provides a general algorithm class, Algorithm/Algorithm, which calls the code for anonymization algorithms, such as the main components of ngram2.0 which is referred to as TrajAnon in the diagram. Note that part of the preprocessing of ngram2.0, namely the universe creation, is in the Common/FileIO class. This package also contains an Algorithm/Common package which contains common data structure code for algorithmic code. The latter provides code for validation, namely count query utility results. Note that FPSM mentioned in Chapter 6 is currently implemented separately. All of these packages use classes in the Common and Lib packages. The former provides common file IO (input/output) and data classes. The latter provides code from other publishers used to accomplish tasks throughout other packages.



Run
TrajAnonymizer
+ (Command line interface)
TrajConfiguration
+ (dat file paths)
+ (field names in CSV file loc, time, id, etc)
+ algorithmType : string
+ algoParams : vector<double>
+ checkIfValidConfig() : bool
Logging
+ getTimeFromExecution() : double
Validation
Validation
+ run(raw : TrajDB&, anon : TrajDB&, universe : vector<Event>*, timeFmt : const char*, maxQuerySize : int)
ValidationIO
+ writeSequencesToFile(seqs : vector<string>, filename : string)
Algorithm
Algorithm
+ Config : TrajConfiguration*
+ run(tdb : TrajDB&, universe : vector<Event>*) : TrajDB*
TrajAnon
- e : double
- lmax : int
- nmax : int
+ run(tdb : TrajDB&, universe : vector<Event>*) : TrajDB*
(Other Algorithm)
Commom
ExplorationTree
- tree : map<string, TreeNode*>
+ levelSet(level : int) : vector<TreeNode*>
+ addNode(node : TreeNode*) : void
+ removeNode(node : TreeNode*) : int
+ createTrajToNode(Node : TreeNode*) : TrajData*
TreeNode
- count : int
- countNoisy : double
- budget : double
- budgetLeft : double
- theta : double
- level : int
- event : Event*
locStrings : vector<string>
- type : int
- released : bool
+ getSequenceKey() : string
Commom
FileIO
+ readCSV()
- convertTime()
+ readSeq()
+ readConfig()
+ writeToSeq()
+ writeToCSV()
TrajDB
- trajdb: map<string, TrajData*>
+ addTrajectoryForID
Event
- loc:Location
- time:time_t
+ getIndividual SequenceKey()
Location
- x:int
- y:int
- APname:string
TrajData
+ traj:vector<Event*>
+ push_back(event: Event*): void
+ insert(pos:int, event:Event*): void
+ pop_back(): void
APEvent
- loc:Location
- time:time_t
- MAC:string
- PID:string
+ getIndividual SequenceKey()
Lib
leanmeanoptionparser
fi
+ (Fi::Fixed <32,16,Fi::SIGNED> type)
ISAAC-64-c++
+ QTIsaac<16, ISAAC_INT> randNumbGenerator
jsoncpp
PicoSHA2
+ hash256_hex_string(hex : string)

Figure 5.1 C++ classes shown in Unified Modeling Language (UML) like format

Figure 5.1 gives a visualization of the overall code structure. The next figure, Figure 5.2, provides a generalized data flow diagram detailing the program execution path. The user interface is a command-line interface (CLI), which is housed in Run/TrajAnonymizer. The code controlling this receives the input file. This input file is transferred to the ngram2.0 code (Common/FileIO and Algorithm/TrajAnon) which anonymizes the data and outputs a synthetic database. Next, the validation code (Validation/Validation) runs counting queries on the data to measure utility. Finally, command is returned to the user via the CLI.

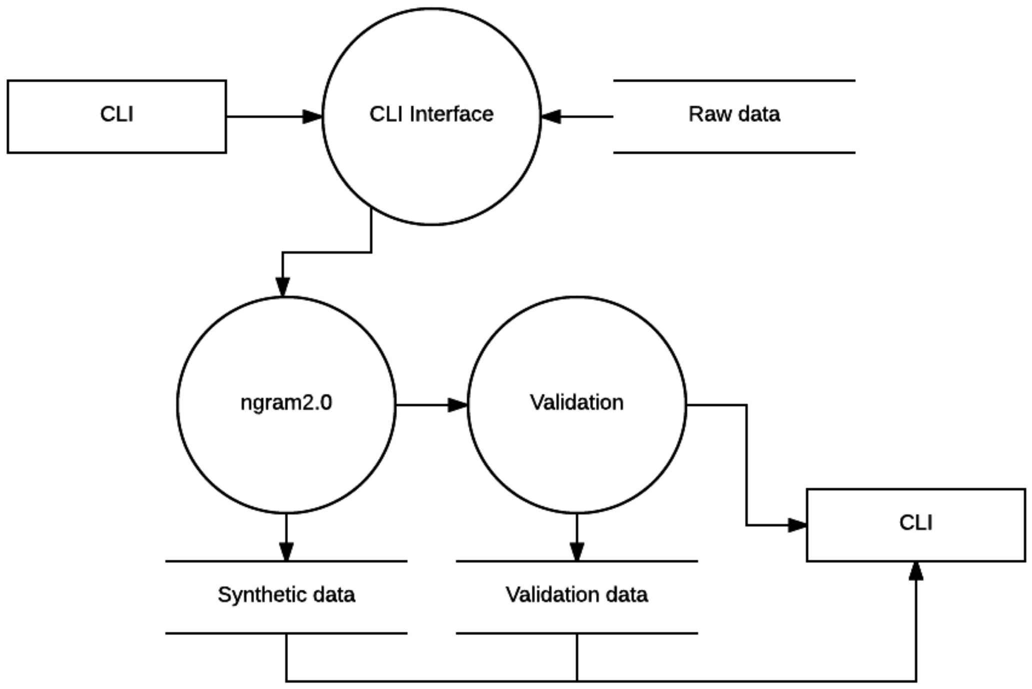


Figure 5.2 Data Flow Diagram (DFD) for ngram2.0

Figure 5.1, provides the overall code structure. This is further explained in the next section.

5.1 Framework

The purpose of the code structure in Figure 5.1 is to create a software framework in which future anonymization algorithms can be implemented. The idea is that the additional code for any new algorithm simply needs to be added to the Algorithm package. Then with this quick addition, these algorithms can be tested and efficiently validated against utility requirements. This quickens the pace that data can be disseminated. The overall process of releasing data can be generalized as:

Generated → Collected → Anonymized → Disseminated

The data is generated by users, computer programs, etc. and is then collected by big data ingestion software. It is then anonymized with the code in this framework, and then securely and selectively disseminated to researchers or others with need.

In reality, ngram2.0 ended up touching various other files, notably including the Common/FileIO and Run/TrajAnonymizer files, and involved a few parameter updates elsewhere. It is left up to future work to finish this framework design to allow fast prototyping of future anonymization algorithms.

The overall framework provides an easy structure to implement anonymization algorithms. The rest of this chapter will explain more in depth how ngram2.0 is implemented in this code structure, based off the algorithm given in Chapter 4. This mainly focuses on part of the Common/FileIO and the whole Algorithm/TrajAnon class since that is where the main ngram2.0 code is written.

5.2 Data Preprocessing

The Data Preprocessing step is relatively straightforward to implement. It is a simple file IO operation reading in a textual columnar CSV file, such as an authmgr log file, or a textual sequential file, such as the MSNBC sequence dataset. Then, this data is stored in memory as a vector of Events.

If the file is a CSV file, there is further processing required. Namely, the trajectories must be constructed. As each row is read in, the event is added to a trajectory with all events matching the unique identifier combination, or a new trajectory is created for that unique identifier combination. For example, all connections (rows) having the same MAC address and username are formed into a single trajectory. Note that this process assumes that the rows in the file are time-ordered based on the timestamp column since every entry with the same ID combination is added to a trajectory as it is being read in from the file.

As with [7], this does not deal with the possible presence of ping-pongs that occur when devices associate/disassociate quickly with many nearby APs. That means there could potentially be a very long trajectory where the user actually staid in one place. It is left to future work to analyze such patterns and their effect on the utility of anonymizing such data.

Creating the universe of items is done at the same time as reading in the file. This process significantly slows this first step down since every spatiotemporal point must be checked against the current universe list. Various comparison options were attempted. This included using the `std::set` structure, but a vector of Events using the `std::find` algorithm was found to provide a similar speed while maintain simple implementation. The largest speed increase was using a `vector<Event>` instead of a `vector<pair<Location, time_t>` structure, meaning the custom `Event` class was faster than using the standard `pair` structure.

The last step of preprocessing is counting n-grams. Since the n-gram counting is not discussed in ngram, this step is placed as part of the Common/FileIO code instead of the Algorithm/TrajAnon ngram2.0 code. It is a simple iteration of the trajectories as discussed in Section 4.3.1, and is implemented as a lookup in a map of n-grams. The map is a `std::map<std::string, int>` structure that holds the n-gram as a string and its count as an integer.

The data is now stored in a Common/TrajDB structure ready to be ingested by the main ngram2.0 code. The next section describes the first main step of the algorithm.

5.3 Exploration Tree Creation

The exploration tree creation step is implemented following Algorithm 4.3 in Section 4.4. From that information, it is relatively straight forward to implement. The only addition is that the total tree height is tracked. As throughout the code, vectors have been chosen to store lists of nodes because of their dynamic nature and ease of use. The following sections provide details on implementation decisions for the three steps outlined in the subsections of Section 4.4.

5.3.1 Computing θ

The math supporting the calculation of θ reduces to a simple formula for its calculation. This is Equation (4.1 given in Section 4.4.1. Note, as Algorithm 4.3 details, this threshold value is computed at the beginning of the Fill Exploration Tree phase on the current node. That value is then used at the end of the Fill Exploration Tree phase to compare against the counts of and set the budgets of the child nodes being added to that current node. The next step will discuss this budget computation.

5.3.2 Computing ε

Computing ε is done according to a function that follows Equation (4.5) in Section 4.4.1, unless the node's level is less than 2. Then, $\varepsilon / n_{\max}$ is returned. Finding $P_{\max}$ is the largest part of this step and is done according to Equation (4.5). The nodes used to calculate $P_{\max}$ are the children of the current node's Markov parent. Calculating the Markovian parent will be discussed in Section 5.4. A simple loop sums these nodes and finds the node with the largest noisy count. So $P_{\max} = \frac{largest\ count}{sum\ counts}$. Then, the resulting budget value is computed as Equation (4.5). Note that all these calculations are done with double values. Once this budget value is computed, Laplacian noise can be added in proportion to its value.

5.3.3 Adding Laplacian Noise

As with many implementations, theory is often hard to implement based on the software and hardware provided. In order to avoid the bit pattern issue that using floating point values introduces, namely allowing gathering information about input values [72], fixed point variables were used in Laplacian calculations as Narayan et al. did [76]. The library chosen to do fixed point arithmetic is one written by McFarlane which has been proposed to the C++ Standards Committee ISOCPP [77].

In order to draw noise from the Laplace mechanism, the code follows the inverse sampling method to draw from the distribution function given a source of randomness R evenly distributed over $[0,1)$ [72].

$$Y \leftarrow sign(r) * \lambda \ln(1 - 2|r|)\,, where\; r = R - 0.5$$

The problem that McFarlane addressed is that the distance between two floating point values varies with relation to the exponent. We set the terms λ and the result of $\ln(1 - 2r)$ to both be fixed point variables. This is then added to each node count as an integer. Here, λ is set to $\Delta f/\varepsilon$, where $\Delta f = \ell_{\max}$ since noise is added to a count query which is affected by whether a whole trajectory is in the original dataset or not, so the sensitivity is bounded by the maximum allowable trajectory length. The technique used by He et al. was considered, of bounding the influence of a trajectory to 1 by making the weight of every transition equal 1/L where L is the number of transitions in the trajectory being added [52]. In the ngram2.0 case, this would have to be 1 / $\ell_{\max}$, which would decrease the noise added but would also decrease the number of nodes that pass θ so it has not yet been implemented in ngram2.0.

For the random number generation, we are using ISAAC created by Jenkins. This is a fast cryptographic random number generator [78].

The form of implementing the Laplacian noise addition was a section that required a bit of further research. This is shown in the design decisions just mentioned. The next phase of ngram2.0 is enforcing consistency constraints on the noisy tree just created.

5.4 Enforce Consistency Constraints

The next phase is making sure the noisy exploration tree resembles the structure of an organic tree created from raw n-gram data. The overall code structure follows that of Algorithm 4.6. This step is necessary since adding random noise adds inconsistencies to the exploration tree since the noise does not follow the same pattern as n-grams do.

The largest logical step was taking the logical steps of Algorithm 4.5 into the code represented in Algorithm 4.6. The logic behind this was discussed in Section 4.5. From Algorithm 4.6, the logic is straightforward to implement. Note again that when children are split into two groups, they are split based off the `released` flag and not if they pass the threshold θ. The one step that requires explanation is the process of calculating the Markovian parent of a node.

Calculating the Markovian parent of a node takes on an iterative process. Remember from Section 4.5 that the Markov parent is the node with the longest trajectory that can be used to predict the current node. The algorithm is as follows. If the current node is at level 0 or 1, return the root since in the former case the node is the root and in the latter case the only parent node is the root. Otherwise, take an iterative approach to finding the Markov parent. Starting with the n-gram location string, iterate while there is no Markov parent found and the new n-gram location string has events in it (length is greater than 0).

During each iteration, pop off the first event element of the string. If a node exists in the exploration tree with this new trajectory and at least one of this node’s children are released, that is the Markov parent. Otherwise, continue searching. Note that checking for released children makes sure this node is a parent of a valid node, though this check may not be necessary.

Implementing the consistency step was rather straight forward. The next and final phase, constructing the actual synthetic database, requires a bit more explanation.

5.5 Construct Synthetic Database

The last phase is constructing the synthetic database from the consistent, noisy exploration tree. This is implemented as described in Algorithm 4.8. The first step is to floor all the noisy counts. The second step is to extend this tree. Thirdly, two synthetic databases are created.

The floor step is done since the counts need to be integers for whole number trajectory counts. If any node has a count ≤ 0, that node is removed, being at that point nonexistent.

The extension step is implemented as an iterative loop over the longest n-grams. First, the nodes in the $levelset$ of the current height of the tree are placed in a vector. Then as long as there are n-grams in this vector, compute if any of these sequences are joinable per Section 4.6.This is done by iterating through this vector, computing if each node’s corresponding g_2 sequence (the node’s sequence minus the first event) is in the exploration tree, if it is then adding all children of the g_2 sequence in a new longest sequence vector, and computing the counts for these new sequences. Note that the children are only added if their computed count is greater than 0. The comparison is done by comparing their generated location strings which is simply the string representation of the n-gram associated with each node. For example, the following could be a location string: `{"Time": "2017-09-01T15:00:00Z", "Location": "LIB-1234"}, {"Time": "2017-09-01T15:00:00Z", "Location": "LIB-2345"}, {"Time": "2017-09-01T15:00:00Z", "Location": "LIB-3456"}.`

Now that the extension step is complete, the synthetic database can be created. The last step in the Algorithm/TrajAnon file is to create a Common/TrajDB structure based on the noisy, consistent, extended exploration tree. This step iteratively adds trajectories to this structure by adding the longest sequences in the tree, subtracting the supporting n-gram counts, then moving on to the next longest sequences until all are added. Note that multiple copies of the same trajectory could be added, as long as the noisy count is greater than 1.

The Run/TrajAnonymizer file then outputs two files. The first is a JSON formatted sequence file, which is outputted with the following format (note that in the actual output file the JSON is in a condensed form with less whitespace):

```
{
   "Trajectory DB":[
      {
         "ID":1,
   "Trajectory":[
      {
         "Event":{
            "Time":"2017-07-07T14:00:00Z",
            "Location":"LIB-1234"
         }
      },
      {
         "Event":{
            "Time":"2017-07-07T14:00:00Z",
            "Location":"LIB-2345"
         }
      }
   ]
},
{
   "ID":10,
   "Trajectory":[
      {
         "Event":{
            "Time":"2017-07-07T14:00:00Z",
            "Location":"LIB-6789"
         }
```

```
                }
            ]
        },
        ...
        {
            "ID":9,
            "Trajectory":[
                {
                    "Event":{
                        "Time":"2017-07-07T14:00:00Z",
                        "Location":"LIB-2345"
                    }
                }
            ]
        }
    ]
}
```

This can be converted back into sequences with a JSON parser. The choices of using this format was done because of the richness of the data being used. Some sequence datasets are simply contain a trajectory per line simply separating each location with a space, such as “`1 2 3`”. This type of file can also contain timestamps, such as separated by a colon “`1:12345 2:12346 3:12347`”. With the data shown above, the data could be converted to be in this type of format, but it was concluded that the JSON format would be clearer, and this can also more easily contain unique IDs or other pertinent information that could be added for different data.

The second file created is a CSV file. The step to create this file iteratively deconstructs the trajectories, making each event along the trajectory a new line in the csv file. Note that this does not guarantee that all rows in the CSV file are time-ordered.

This section described the implementation of the last phase of ngram2.0. Namely, it outlined the last editing phase of the exploration tree, and the creation of the two output files. The next section will take a step back to describe overall code structure design decision.

5.6 Conclusions

The previous sections described specific implementation of the ngram2.0 algorithm in the context of the software from Figure 5.1. The implementation closely follows the pseudo code given in Algorithm 4.1, Algorithm 4.3, Algorithm 4.6, and Algorithm 4.8. The most notable implementation change is using a fixed point library for the Laplacian distribution noise computations. Otherwise, these algorithms are translated directly to C++ code. The next Chapter discusses the results from this implementation, focusing on utility results of various input datasets.

6 Results

Ngram2.0 is tested for utility results based on counting queries (CQ) and frequent sequential pattern mining (FSPM). Utility has been measured with various means by previous work. This includes maximum distance between pairs of locations [52], trip distribution in grids [52], frequent patterns [19], [51], [52], average relative error of CQ [19], [51], [79], and complementary cumulative distribution function (CCDF) [41]. These techniques have benefits, including ensuring the resulting trajectories fit the geographical topography. The problem with some of these techniques is that they do not fit with the type of data being used. These data restrictions are described in Section 6.1. Thus, the technique used by Chen et al., and other authors as described above, has been adopted which uses results from counting queries (CQ) and frequent sequential pattern mining (FSPM) to compare the resulting synthetic database with the original raw database. The former measures the number of matches to a specific query, while the latter measures the amount of statistically relevant patterns of a dataset. The main contribution of this chapter is to provide an analysis with these techniques of how data density/sparsity affects the utility of the output anonymized dataset from a privacy algorithm.

A counting query is simply a query that counts the number of occurrences of a location or trajectory in the dataset. The measurement follows the work of previous authors [51], [80], [81]. The utility is measured by the relative error of the answer on the sanitized database $\widetilde{\mathcal{D}}'$ with respect to the true answer on the original raw database $\mathcal{D}'$:

$$error\left(Q\left(\widetilde{\mathcal{D}}'\right)\right) = \frac{\left|Q\left(\widetilde{\mathcal{D}}'\right) - Q(\mathcal{D}')\right|}{\max\{Q(\mathcal{D}'), s\}},$$

where s is a sanity bound used to mitigate the effect of queries with very small selectivities [51], [80], [81]. Thus, if there is an excessively small query result on the original dataset, this sanity bound is used instead, mitigating the effect of any excessively small counts. As per [51], [80], [81], s is set to 0.1% of $|\mathcal{D}'|$. The code to do this is integrated into the Verification section of ngram2.0. This type of counting query can be used as the foundation for many calculations, such as finding the amount of devices using a wireless network or more specifically a wireless access point.

FSPM is a more specific data mining task. Given a positive integer K, this calculates the top K frequent subsequences in $\mathcal{D}'$ and $\widetilde{\mathcal{D}}'$, denoted by $F_K(\mathcal{D}')$ and $F_K(\widetilde{\mathcal{D}}')$, respectively. Once these two frequent sets are calculated, two statistics are calculated. The first is the true positive ratio (TPR), which is the percentage of frequent patterns that are correctly identified, or the number of patterns that are in both the raw and sanitized datasets. This is calculated as [51]:

$$\frac{\left|F_K(\mathcal{D}') \cap \mathrm{F}_\mathrm{K}(\widetilde{\mathcal{D}}')\right|}{K}$$

The second quantity calculated is the utility loss (UL), which factors in the supports of the patterns instead of just their existence. This is given as [51]:

$$\frac{\sum_{F_i \in F_K(\mathcal{D}')} \frac{\left|\sup\left(F_i, F_K(\mathcal{D}')\right) - \sup\left(F_i, F_K(\widetilde{\mathcal{D}}')\right)\right|}{\sup\left(F_i, F_K(\mathcal{D}')\right)}}{K}$$

where $\sup\left(F_i, F_K(\mathcal{D}')\right)$ and $\sup\left(F_i, F_K(\widetilde{\mathcal{D}}')\right)$ are the supports of F_i in $F_K(\mathcal{D}')$ or $F_K(\widetilde{\mathcal{D}}')$, respectively. The supports are the number of occurrences of the pattern F_i in each frequent pattern set $F_K(\mathcal{D}')$ or $F_K\left(\widetilde{\mathcal{D}}'\right)$ (ie in each dataset). If the UL is 0, then $F_K(\mathcal{D}')$ and $F_K\left(\widetilde{\mathcal{D}}'\right)$ are identical, since the support values would be the same. If it is 1, they are totally different, since the numerator sum would equal the denominator K. The MAFIA codebase was used to perform initial pattern counts [82], [83], then custom Java code used to compare the two resulting pattern sets. Note that all frequent patterns of size 1 are removed since they are trivial patterns.

Since this FSPM task provides sequential pattern information about the datasets, it could be used for something like finding high usage paths in a wireless network topology. This information could then be used to influence further new network device installation.

As a summary, Table 6.1 Summary of desired results of utility measures shows the desired values for these quantities. CQ measure the relative error of counting. Error should be low, so 0 is desired. FSPM UL measures the existence of and difference in counts of frequent patterns. Since it is partly a difference of counts, a value of 0 is again desired. FSPM TPR measures the intersection of the existence of frequent patterns. Thus it is a percentage and 100% similarity is desired.

Measure	Undesired	Desired
CQ	1	0
FSPM TPR	0%	100%
FSPM UL	1	0

Table 6.1 Summary of desired results of utility measures

The first step in collecting results is to use the verification code on the input and output of ngrampy to provide a baseline and verify which parameters should be used for highest utility. This is in Section 6.2. Then Section **Error! Reference source not found.** provides verification analysis of ngram2.0. First, MSNBC data is used to verify that the utility results match that from ngrampy. Then, the parameters seen to provide the most utility from the first verification step are used for verification of authmgr and edited authmgr data for ngram2.0. The last two sections of this results chapter, Sections 6.4 and 6.5, provide analysis of the algorithm itself. The former provides analysis of runtime complexity. The latter provides privacy analysis based on differential privacy guarantees.

6.1 Data

There are two datasets used to analyze this algorithm. This includes a sequential and a tabular database. Their main features are detailed in Table 6.2.

Dataset	Num Unique Events	Traj Count	Universe Size	Mean Traj Len	Max Traj Len	Min Traj Count	Mean Traj Count	Max Traj Count
MSNBC	4698794	989819	17	4.7	14795	16972	276399	940469
authgmr	6400000	57000	5800	100	16000	1	1000	40000

Table 6.2 Characteristics of the two input datasets (MSNBC and authmgr)

The first dataset is MSNBC [84]. This is one of the two datasets used in [51] and contains URL visitation information from msnbc.com and is publically available via the UCI Machine Learning Repository. Specifically, each event in the sequences correspond to page requests such as “frontpage”, “news”, or “sports”. Due to the small universe size comparative to the large number of sequences as shown in Table 6.2, this is a relatively dense dataset. This makes it ideal for the n-gram algorithm. It does not contain explicit timestamps but is instead time-ordered location points. Thus it does not use the full capacity of ngram2.0 but is used to test the algorithm to ensure it returns similar utility results to ngrampy.

The second dataset is authmgr. This is a dataset of authentication manager data over a period of 24 hours from Virginia Tech Information Technology. The day chosen was a weekday with classes in session to ensure high network usage. This contains rows of user device associations with access points (APs) in the Virginia Tech Eduroam wireless internet network. Note that all de-authentication entries were removed since they did not contain AP names. Each row contains a: timestamp, access point name, MAC address, and PID. The MAC, or media access control, address identifies a specific device, whether that is a smartphone, laptop, or similar device. The PID, or Personal Identifier, identifies the specific user connecting to the network. Thus these two pieces of identifying information combine to form a unique device/person identifier pair. These rows are combined to form trajectories' spatiotemporal points. It has a much larger universe size in comparison to the number of sequences, as seen in Table 6.2 which makes the data much sparser. This dilemma is addressed in Section 6.3 while computing utility results by making the data less sparse while maintain essential sequential information. This is the dataset that ngram2.0 was designed for. It is also not publically available so identifiers are hashed (with a salt) before use to ensure at least some level of privacy protection during analysis.

It is obvious that the application of this algorithm produces different results for MSNBC data than for authmgr data. The main reason for this difference is that the authmgr data is much sparser than the MSNBC data both in the sense that each location has much fewer hits and each trajectory may have less points along its path because of lack of APs, whereas every location visited is recorded in the MSNBC data.

The problem is that the ngram algorithm makes an implicit assumption that there are a large number of counts for each n-gram, which is not true with many applications [54]. Thus a simplified version of the authmgr data is used to provide a high level of utility results.

The next section will use the MSNBC dataset to compare the utility results from both ngrampy and ngram2.0 to ensure they produce similar results for the same dataset. It will also verify the ideal parameter values to be used for ngram2.0, except for the parameter t since that is not used by ngrampy.

6.2 Validation

This section will discuss how to choose what values to use for the 4 parameters: ε (privacy budget), $\ell_{\max}$ (maximum trajectory length), $n_{\max}$ (maximum n-gram length), and t (time bucket). The privacy budget is the differential privacy parameter so its recommended value is based off previous recommendations by Dwork and others in the area. The last three parameters are algorithmic parameters. The first two of these were introduced by Chen et al. while the last was introduced in this thesis for this algorithm.

Previous researchers have recommended values for some of these parameters. Generally Dwork suggests using a relatively small ε value, ranging from 0.01 to 3 [17], though she states that the choice is a social question. This thesis uses values of 0.1 and 1, but this choice should be further analyzed by future research on differential privacy. For the more algorithmic parameters $\ell_{\max}$ and $n_{\max}$, Chen et al. recommend values of 20 and 5 respectively, though it is unclear why. Time sensitivity, or the time bucket, is a new parameter added by this research endeavor.

In order to verify the first three parameter value choices, the first results taken were from the original ngrampy code run with MSNBC data. This was taken as a baseline and to verify the best parameters to use for highest utility, at least for that dataset. There were 32 combinations of parameters run, namely ε of 0.1 and 1; $\ell_{\max}$ of 5, 10, 15, and 20; and $n_{\max}$ of 2, 4, 5, and 7. The results were very close with each run, but a few trends did emerge.

The CQ results can be viewed in Figure 6.1. The trend is the three groupings of lines. The top grouping is all the trials with $n_{\max}$ of 4 or 7, the middle $n_{\max}$ of 2, and the bottom $n_{\max}$ of 5. Thus any of these 32 combinations with $n_{\max}$ values of 4 or 7 produce slightly higher utility. Within this group, the choice of the other two parameters did not seem to greatly affect the CQ utility level.

The FSPM results for TPR are given in Figure 6.2. There are two groupings of the trials. The top grouping contains all the trials with an $n_{\max}$ of 5, while the bottom grouping contains the rest, namely $n_{\max}$ of 2, 4, or 7. Thus this slightly contradicts the CQ results.

The FSPM results for UL are given in Figure 6.3. This shows the same trend as the TPR results. Note again for UL that a lower value is better. Thus, an $n_{\max}$ of 5 produces the best UL utility result.

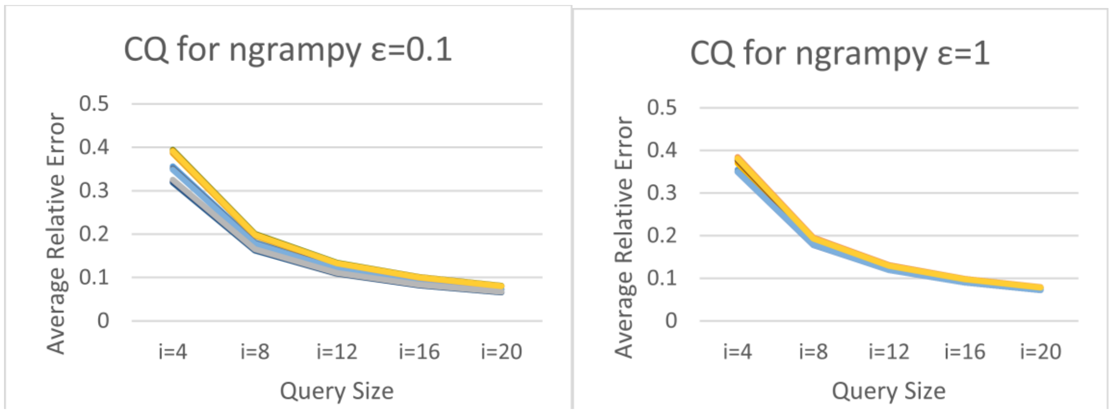


Figure 6.1 Count query results for MSNBC data run through ngrampy with varying parameter values

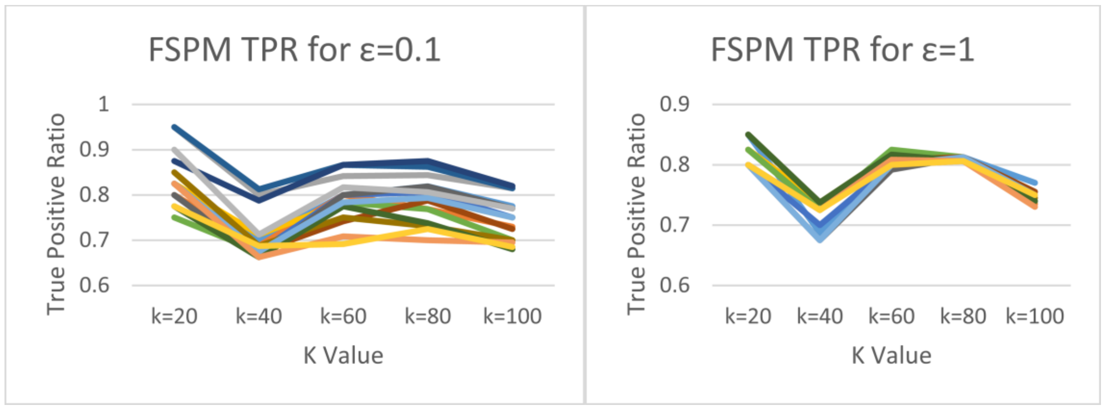


Figure 6.2 FSPM true positive ratio results for MSNBC data run through ngrampy with varying parameter values

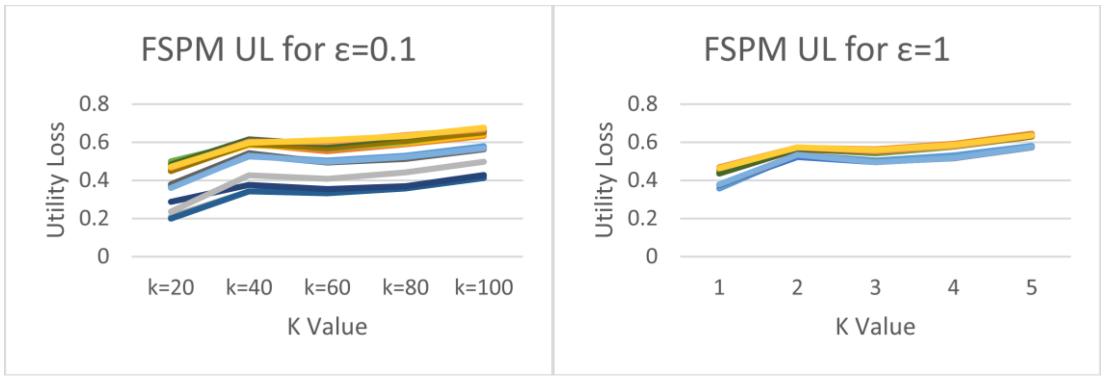


Figure 6.3 FSPM utility level results for MSNBC data run through ngrampy with varying parameter values

From these trials, no strong conclusion can be made. Any grouping of the parameters used result in a similar utility level. However, some trends are present. The CQ results point toward using an $n_{\max}$ of 4 or 7, while the FSPM results suggest a usage of an $n_{\max}$ of 5. Thus we conclude that for a combined maximal utility level, an $n_{\max}$ of 4 or 5 is best, while it does not matter what is chosen for the other values as long as they are within the range used. Thus, further trials would benefit in using an ε of 0.1 or 1, $\ell_{\max}$ of one of the four options, and $n_{\max}$ of 4 or 5 based on the MSNBC results.

To verify accurate working of ngram2.0, the MSNBC data was run through this code with parameters ε of 0.1 and 1; $\ell_{\max}$ of 5 and 20; and $n_{\max}$ of 2, 4, 5, and 7. A subset of this data is shown in Figure 6.4, Figure 6.5, and Figure 6.6. These figures show that ngram2.0 outputs approximately the same utility results as ngrampy for the same data. The differences displayed in these graphs could be due to the number of trials run, specific noise mechanism implementation, or other slight algorithmic difference.

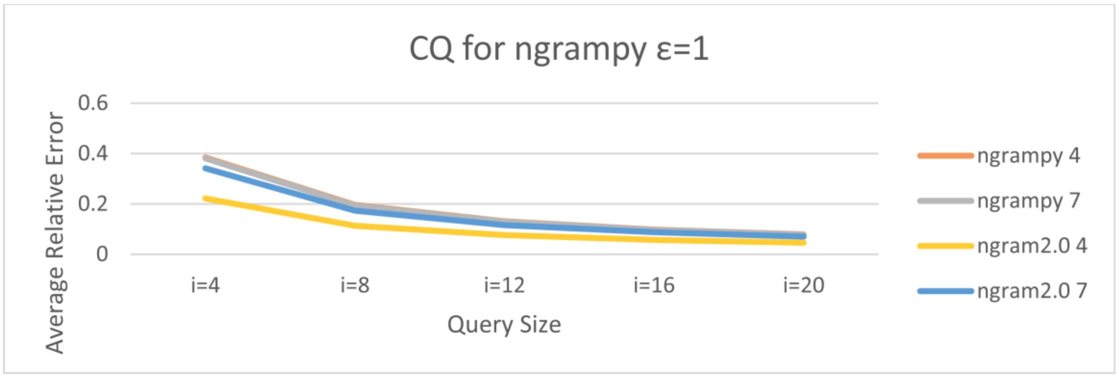


Figure 6.4 Count query results for MSNBC data run through ngram2.0 with varying parameter values

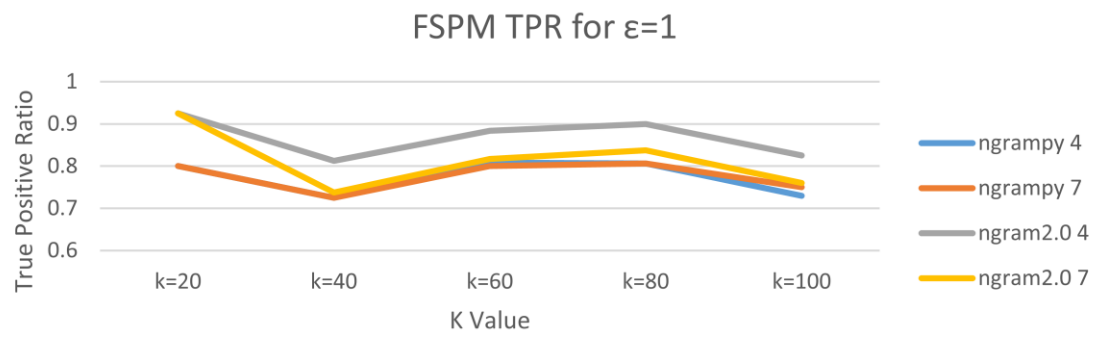


Figure 6.5 FSPM utility level results for MSNBC data run through ngram1.0 with varying parameter values

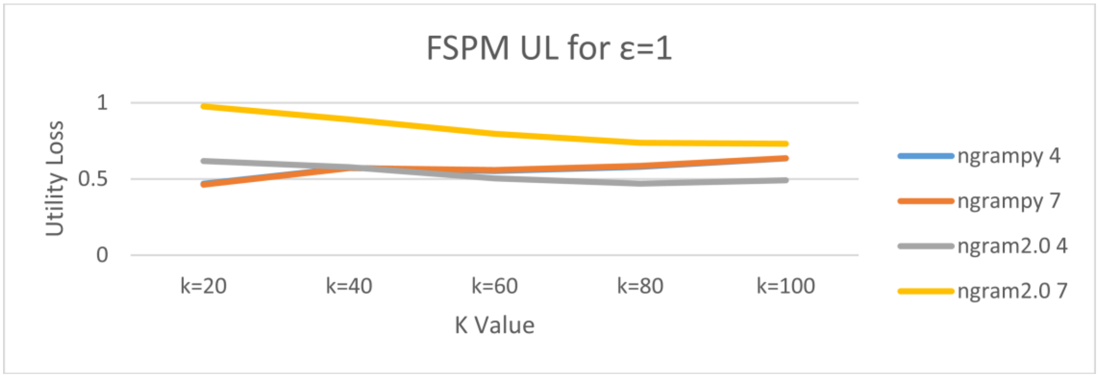


Figure 6.6 FSPM utility level results for MSNBC data run through ngram2.0 with varying parameter values

From the matching results of these two programs, the same conclusions can be made from MSNBC run through ngrampy. Specifically, further trials would benefit in using an ε of 0.1 or 1, $\ell_{\max}$ of 20 or one of any of the four options, and $n_{\max}$ of 4 or 5. The next section will demonstrate utility results based on authmgr data.

6.3 Evaluation

After analyzing the results from MSNBC data with regards to best parameter choice, authmgr datasets were run through ngram2.0. This introduces an additional parameter, the time bucket sensitivity or sensitivity for short. It was intended to use sensitivities of $t = 60, 900, 3600, 21600, 86400$ (or minute, 15 minute, hour, 6 hour, day). This would have added 5 times the amount of trials, but it was found that very small sensitivities resulted in extremely poor utility so a smaller subset of parameter choices was used. This section will detail the utility results and what was done to the authmgr data to make it more performant in the ngram2.0 algorithm.

We found that using the raw authmgr dataset resulted in very poor utility using ngram2.0. This was because of the data characteristics described in Table 6.2 and Section 6.1. Namely, the amount of universe items compared to the number of trajectories (or number of unique entries) in the authmgr data is small compared to that of MSNBC. Thus, there are many APs with a low number of user connections, or spatiotemporal points with low counts, compared to a high number of overall APs, or a large universe. An aggregation of authmgr logs over more than one day, say a week, could raise utility levels. The problem is that this would take out any temporal aspect. Thus, authmgr logs over a longer time period would actually not help utility if a maximum time sensitivity of one day ($t = 86400$) is used. This large universe size means a greater amount of noise needs to be added which greatly distorts the comparatively low count spatiotemporal points. Thus, the authmgr dataset was edited in various ways to reduce the universe size and increase the mean count of trajectories to reduce the amount of universe locations with extremely small counts. Table 6.3 shows these changes. Note the mean and max trajectory lengths stayed almost constant since the universe locations were generalized while the trajectories themselves left intact. The first line, "authmgr" is the original dataset. The next line is this dataset except the access points are generalized from unique access point names to building names. This removes precise room location statistics, but maintains general location throughout the campus. The last four lines use the "authmgr bldgs" dataset as a base but then removes every entry that corresponds to buildings that have small counts. For example, in "authmgr bldgs. 1,000" all buildings, and thus all entries with a connection to this building, with counts less than 1,000 are removed.

Dataset	Num Unique Events	Traj Count	Univ-erse Size	Mean Traj Len	Max Traj Len	Min 1-gram Count	Mean 1-gram Count	Max 1-gram Count
authmgr	6400000	57000	5800	100	16000	1	1000	40000
authmgr bldgs	6400000	57000	150	100	16000	5	41000	450000
authmgr bldgs 100	6400000	57000	140	100	16000	100	43000	450000
authmgr bldgs 1,000	6300000	57000	130	100	16000	1000	49000	450000
authmgr bldgs 10,000	6100000	54000	80	100	16000	10000	76000	450000
authmgr bldgs 100,000	3700000	39000	80	90	16000	100000	188000	450000

Table 6.3 Approximate characteristics of the version of the authmgr two input dataset

Figure 6.7, Figure 6.8, and Figure 6.9 show the utility results of these various dataset changes. Since these datasets are very similar, there is not a whole lot of difference. But in general, the datasets lower in Table 6.3 result in higher utility levels in these three graphs.

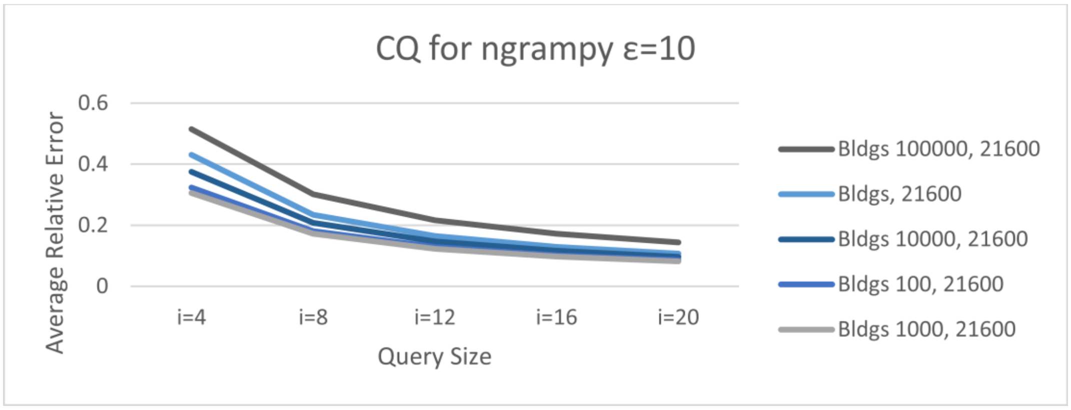


Figure 6.7 Count query results for authmgr data variants run through ngram2.0 with ε=10, ℓmax=20, nmax=5, t=21600

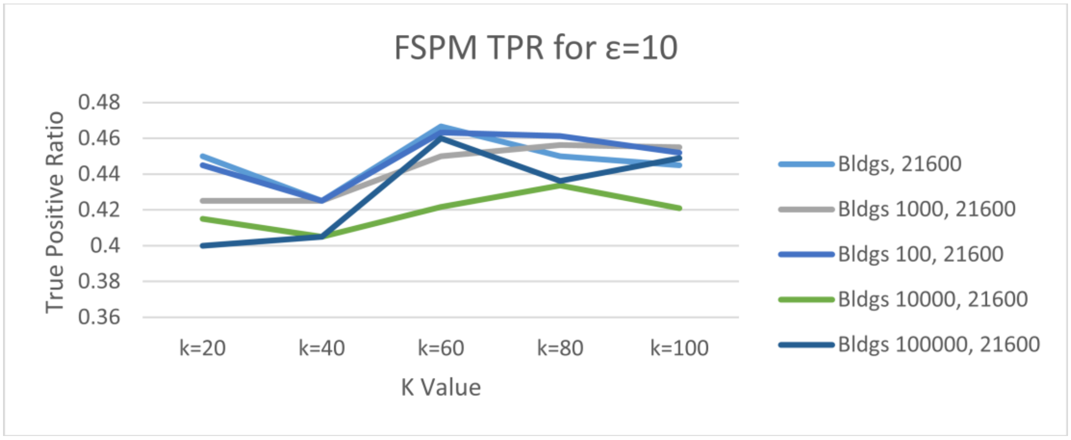


Figure 6.8 FSPM true positive ratio results for authmgr variants data run through ngram2.0 with ε=10, ℓmax=20, nmax=5, t=21600

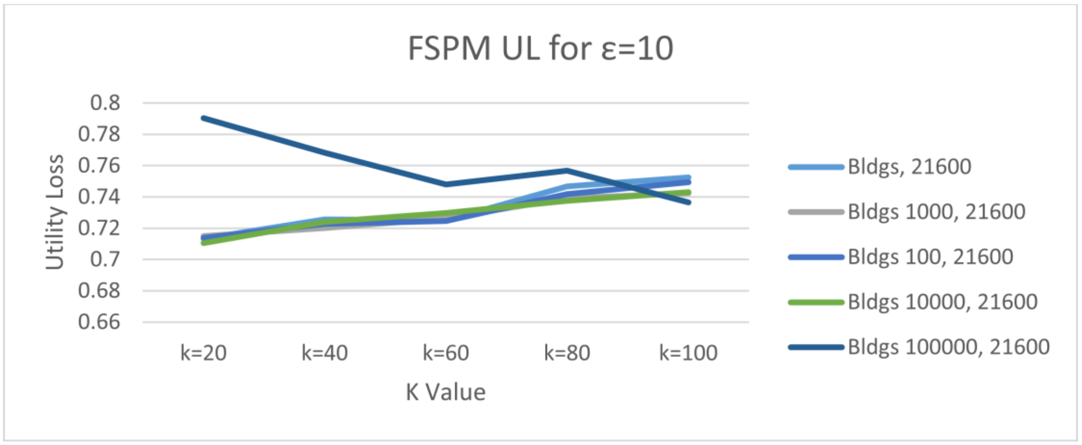


Figure 6.9 FSPM utility level results for authmgr variants data run through ngram2.0 with ε=10, ℓmax=20, $n$max=5, t=21600

From these results, there is a slight bias for best utility from results coming from either "authmgr bldgs 100" or "authmgr bldgs 1,000". Thus, to show different t value utility results, the "authmgr bldgs 1,000" dataset is run with varying t parameter values. These results are shown in Figure 6.10, Figure 6.11, and Figure 6.12. Smaller t results in lower/worse utility results. Figure 6.10 shows that for CQ analysis the difference is not significant, but the trend remains that smaller time sensitivity values lead to better utility. Figure 6.11 and Figure 6.12 shows this as well, but in a more prominent manner. Further trials use a t=21600 value to maintain a level of time sensitivity, while maintaining high utility.

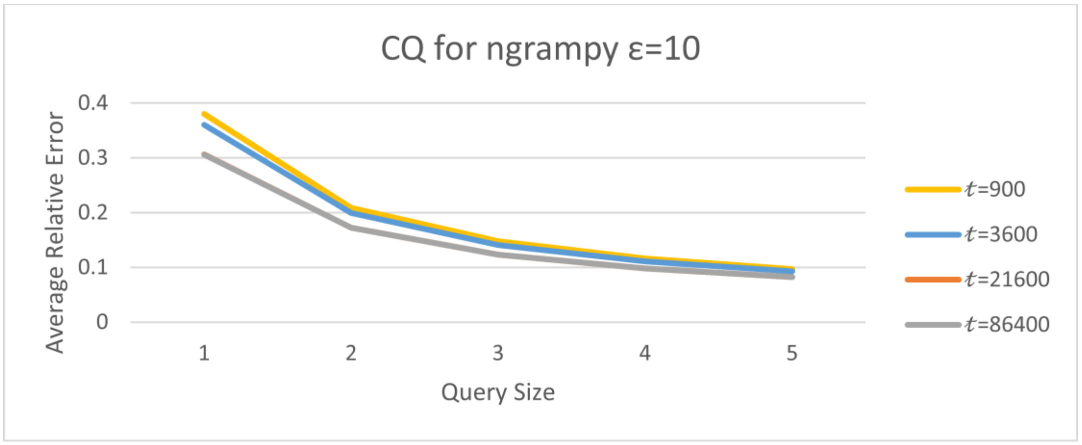


Figure 6.10 Count query results for authmgr bldg 1,000 data run through ngram2.0 with ε=10, ℓmax=20, nmax=5, while altering t

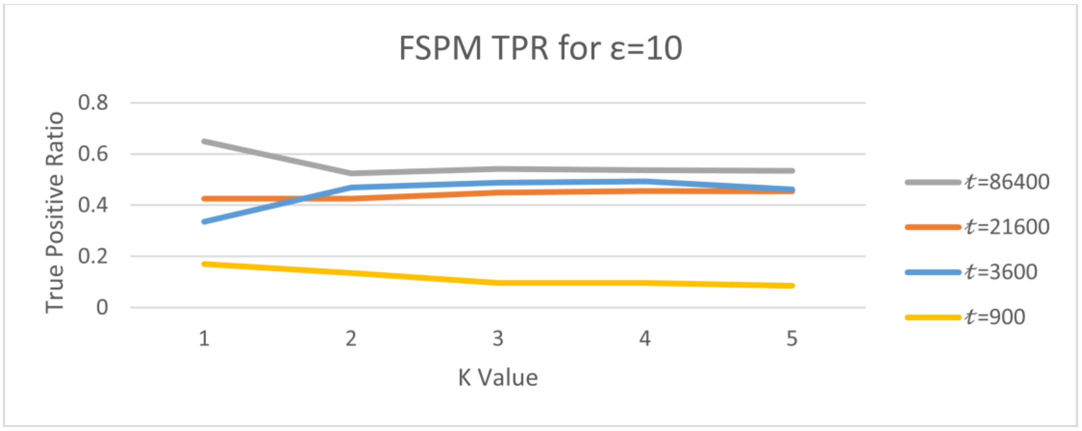


Figure 6.11 FSPM true positive ratio results for authmgr bldg 1,000 data run through ngram2.0 with ε=10, ℓmax=20, nmax=5, while altering t

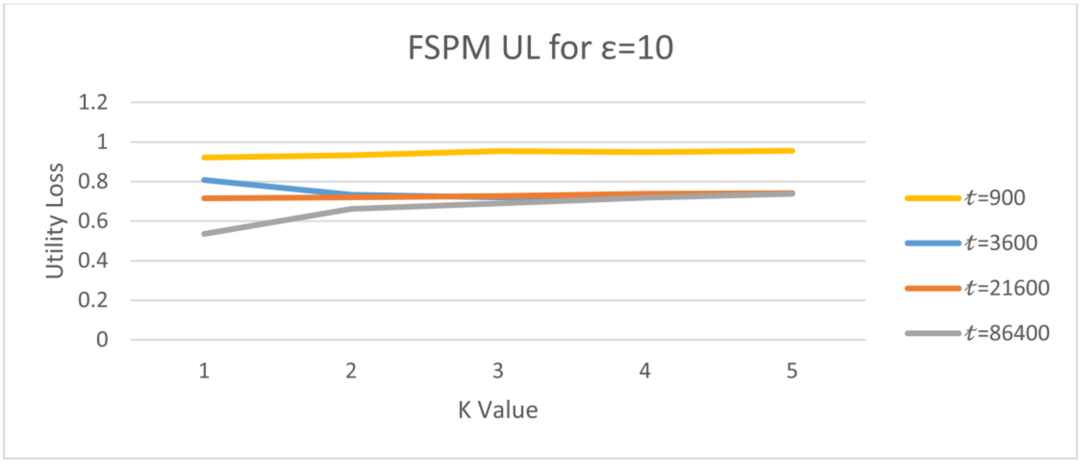


Figure 6.12 FSPM utility level results for authmgr bldg 1,000 data run through ngram2.0 with ε=10, ℓmax=20, nmax=5, while altering t

The final set of graphs gives a summary of the results. It compares the results of MSNBC data and various authmgr variants run through ngram2.0. These are Figure 6.13, Figure 6.14, and Figure 6.15, from which it can be seen that the “authmgr bldgs 1,000” dataset is much closer to the MSNBC dataset results for each graph. “Authmgr bldgs” improves on the pure “authmgr” dataset, which can be seen the most in the TPR graph (Figure 6.14). “Authmgr bldgs 1,000” does not provide exactly the same utility levels as the MSNBC data, but does provide a utility level that is much closer than the other authmgr variants.

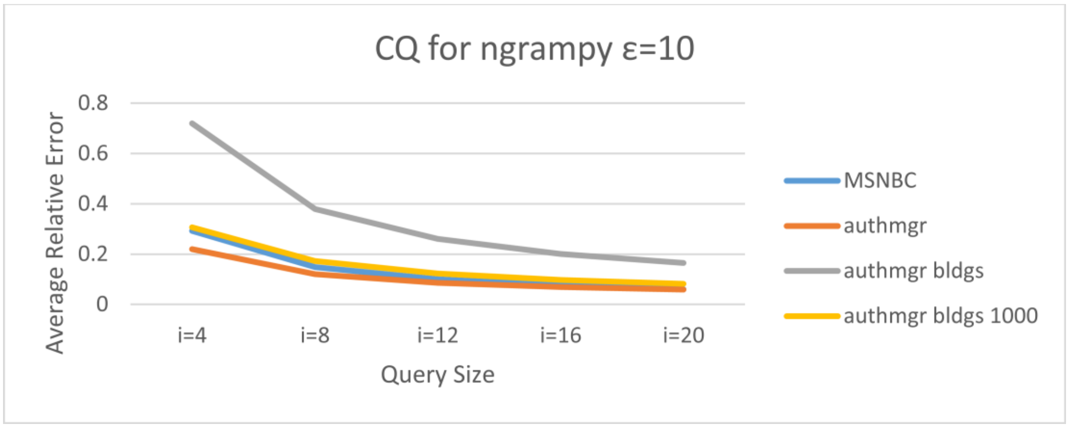


Figure 6.13 Count query results for authmgr data run through ngram2.0 with ε=10, ℓmax=20, nmax=5, t=21600

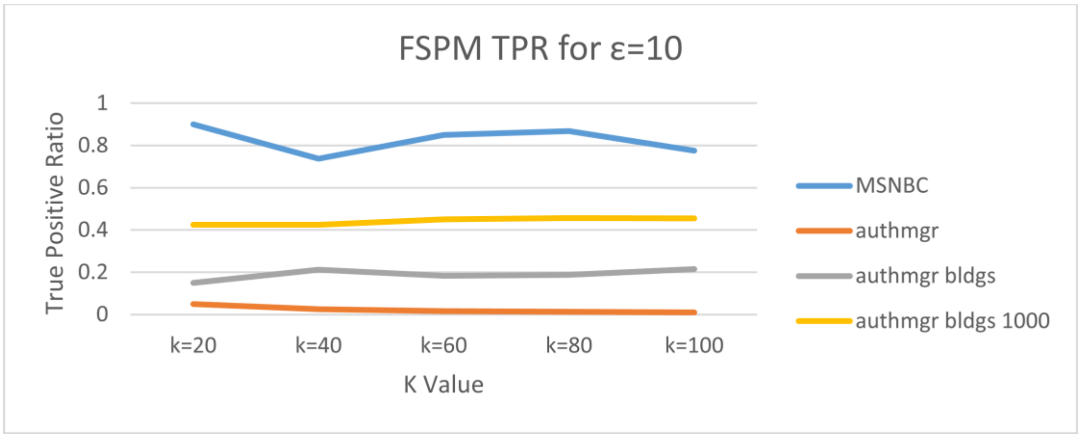


Figure 6.14 FSPM true positive ratio results for authmgr data run through ngram2.0 with ε=10, ℓmax=20, nmax=5, t=21600

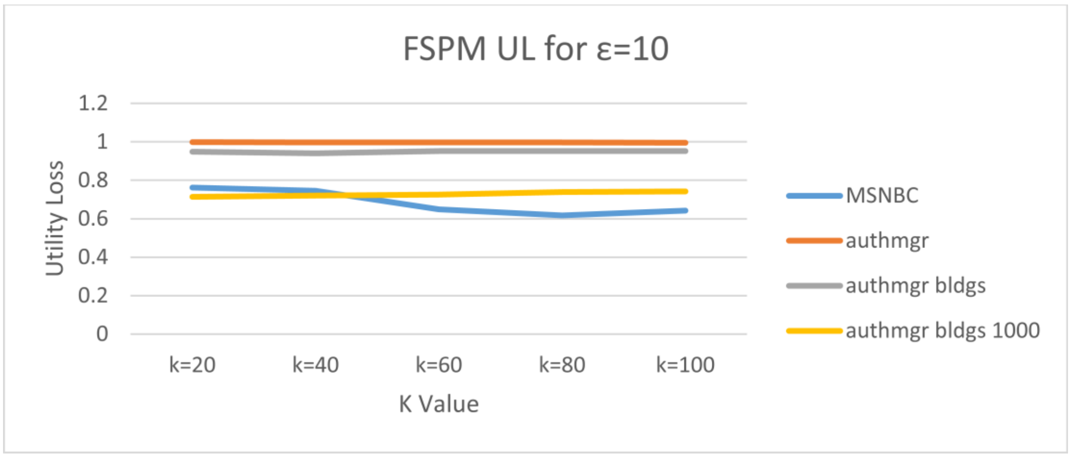


Figure 6.15 FSPM utility level results for authmgr data run through ngram2.0 with ε=10, ℓmax=20, nmax=5, t=21600

This section gave an analysis of the ngram2.0 algorithm with various authmgr dataset versions. It was found than an edited version of this dataset results in utility results approaching that of less dense datasets. The following section compares ngram2.0 in general against previous algorithms.

6.3.1 Comparison

Overall, this algorithm provides a level of utility for the authmgr dataset that comes close to reaching that of previous algorithms. This is show in Figure 6.16, Figure 6.17, and

Figure 6.18. These are the representations of the same utility measures as before, counting query (CQ), frequent sequential pattern mining (FSPM) true positive ratio (TPR), FSPM utility level (UL). This is done for different algorithm and dataset combinations. All the algorithms are run with $\varepsilon = 1$, $\ell_{\max} = 20$, $n_{\max} = 5$, and the MSNBC dataset, except for the line labeled ngram2.0/authmgr which used the authmgr dataset and an additional parameter of $t = 86400$. The algorithms being compared are ngram as implemented by ngrampy described earlier [51], Baseline [51], Prefix [47], and FFS [85]. Baseline is the ngram algorithm without adding noise so it shows the difference added simply by using n-grams to represent the data. Prefix is an algorithm by Chen et al. using a prefix tree structure as described in Section 2.4. FFS is a method by McSherry and Mahajan for finding frequent substrings using a prefix structure. It does not specifically deal with sequential information but provides a similar algorithm since it uses substrings to anonymize the published dataset. Note that the data results from these other algorithms is from [51].

Overall, the ngram2.0/MSNBC combination provides a similar utility level as compared to ngram/MSNBC combination. The differences could be attribute to the number of trials, specific choice of noise mechanism, or other algorithmic choices. The ngram2.0/authmgr combination provides a similar utility level as the other combinations. Comparatively it has the worst utility, but the other combinations use other datasets so it is not a perfect comparison. It just shows that it is relatively close to the others. If the other algorithms could be run with the same data, it is presumed that they would all be more similar to the ngram2.0/authmgr algorithm/dataset because of the differences in the dataset, which will be described in the next section.

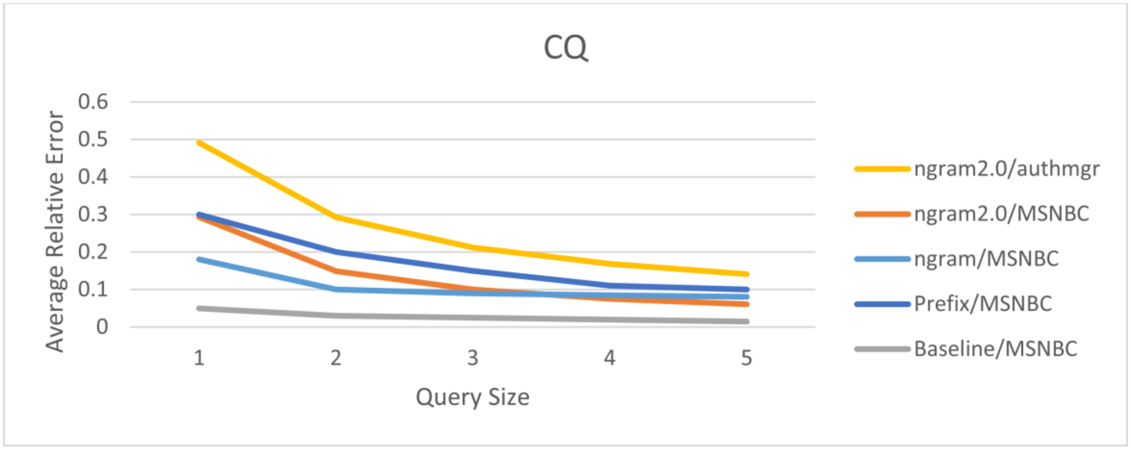


Figure 6.16 Count query results for different algorithms with ε=1, ℓmax=20, nmax=5

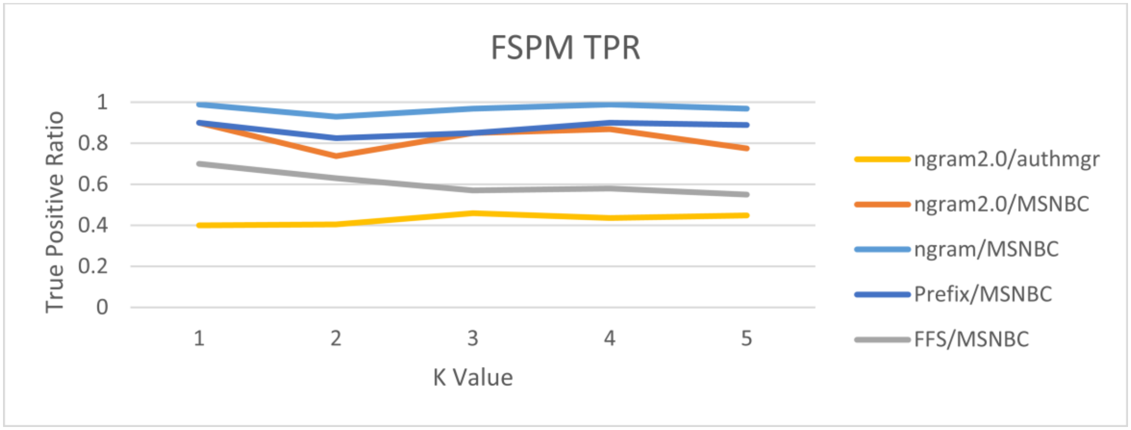


Figure 6.17 FSPM true positive ratio results for different algorithms with ε=1, ℓmax=20, nmax=5

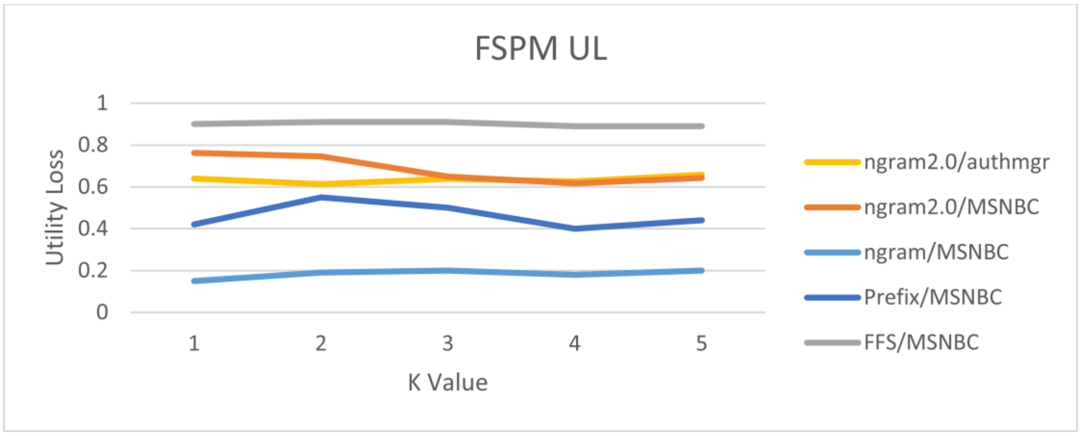


Figure 6.18 FSPM utility level results for different algorithms with ε=1, ℓmax=20, nmax=5

The desired use case for ngram2.0 provides adequate but not extraordinary utility results. The next section will analyze why this may be the case.

6.3.2 Analysis

The above analysis provides the general utility results of ngram2.0. There still remains an open question concerning how to quantify the type of dataset that provides higher utility. From the results described above, it can be seen that data has to have a small universe size relative to the number of counts at each universe item. In order to quantify this, the authmgr data could be viewed as an undirected simple graph. In this graph structure, the vertices could be the APs and devices, while the edges would be connections between these, or the number of hops along the trajectories. Note the choice is an undirected graph, as opposed to directed, since devices connect to APs and not the other way around. With this in mind, the graph density can be calculated with:

$$D = \frac{2|E|}{|V|(|V| - 1)}$$

The problem with this calculation is that since vertices are both APs and devices, a larger amount of devices would bring the density value down, and then presumably the utility level, but larger amounts of devices means more connections and higher utility in the previous calculations. The problem is that we are dealing with trajectories that contain directed information not contained in this density calculation. Thus a new value could be calculated where the vertices are only the number of APs, or universe items. This value would not be bounded between 0 and 1. Thus to not be confused with the previous formula, a new equation is defined here as universe density:

$$D_U = \frac{2|E|}{|U|(|U|-1)}$$

So this value quantifies a ratio of connections over universe size.

The results of this calculation is given in Figure 6.19. This shows the logarithmic relationship of the datasets described earlier vs the output of the above equation. It is concluded that the value of this density ratio must be at least greater than one, though the higher utility started around 1,000. This means that the number of connections between nodes should be at least 100 times the number of universe items, but ideally greater than 10,000 times. It is seen that the utility results only slightly improved with more removal of less sparse data. Based off this graph, the “authmgr bldgs 100,000” dataset would have a greater utility than the other authmgr datasets, but from the results already described previously in this section, this dataset did not have that noticeably improved results. Thus, more testing would have to be done to conclude the extent to which that difference improves utility.

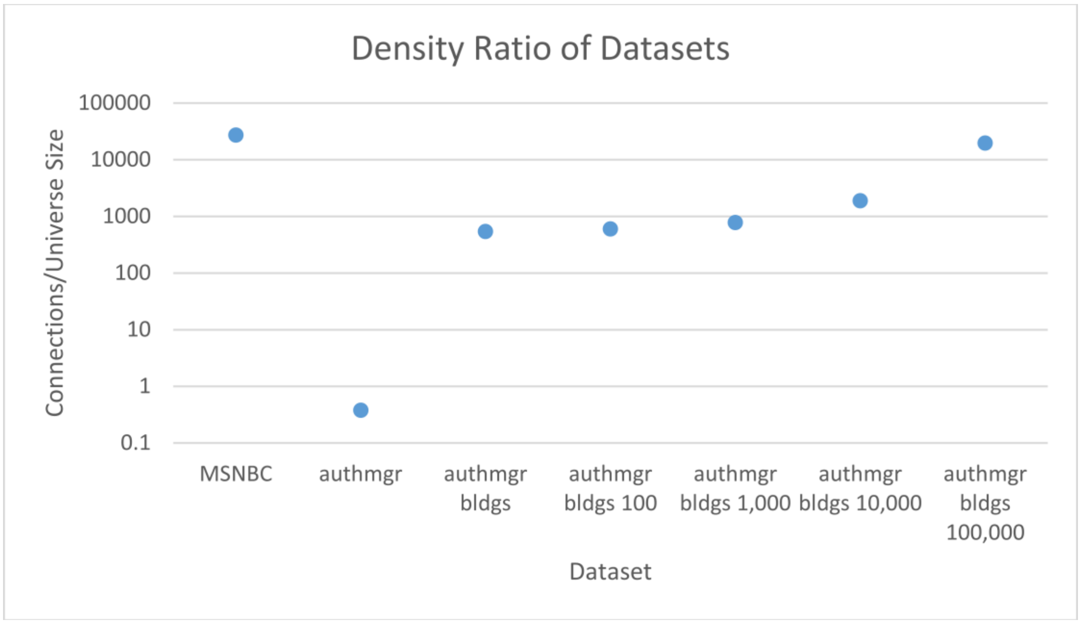


Figure 6.19 Sparsity of input datasets

The next section will discuss algorithm complexity in broad terms. It will mainly focus on time taken to complete different sections of the ngram2.0 algorithm.

6.4 Algorithm Complexity

This section will provide an overview of the complexity of ngram2.0. This is not a full complexity analysis, but a discussion of how input data affects how long the algorithm takes to execute.

A comparison of the time to execute this algorithm is shown in Table 6.4. This table displays the datasets mentioned earlier with the parameters used and time to execute the individual steps of ngram2.0.

The largest time chunk within this algorithm was the Data Preprocessing step which took on average 80-90% of the total time for the authmgr datasets. There are three main parts of this step. First, the file input, which includes reading in the file and checking whether the events are in the universe, took the largest portion at about 65% of the Data Preprocessing time. Second, the remaining 20% is converting data structures, namely converting the row structure into trajectories. Third, the n-gram calculation takes approximately 5% of the Data Preprocessing step’s time. The first part of this Data Preprocessing step takes a significant amount of time because of the need to compare every added event to every universe item. This could be optimized, but that was not the focus of this thesis.

The rest of the remaining time is mostly taken up with the Construct Synthetic Database step. The extension part of this step did not take much time, so the majority of the time is taken by creating the trajectories by iterating over the exploration tree. Like other steps, this involves iterating over all the nodes in the tree, but for every n-level node, n other nodes are also affected, which leads to longer execution time.

The MSNBC dataset did have a shorter Data Preprocessing time because the data was directly read in as trajectories instead of being read in as rows and then creating the trajectories. Though this dataset had a longer Construct Synthetic Database step presumably because of the fact that a greater data complexity is stored in the exploration tree for this data. The smaller universe size leads to this greater complexity since this smaller universe size allows less noise to be added and thus retain more sequential information and thus a taller tree.

Dataset	ε	$\ell_{\max}$	$n_{\max}$	t	Prep	N-gr	Extr	Con	Syn	X
MSNBC	10	20	5	-------	70	60	5	0	70	80
Authmgr	10	20	5	86400						40,000
Authmgr bldgs	10	20	5	86400	230	20	5	1	20	1,000
authmgr bldgs 100	10	20	5	86400	230	20	5	1	20	900
authmgr bldgs 1,000	10	20	5	86400	230	20	5	1	15	800
authmgr bldgs 10,000	10	20	5	86400	220	20	5	1	15	500
authmgr bldgs 100,000	10	20	5	86400	140	15	1	0	15	70

Prep	= Data Preprocessing
N-gr	= N-gram Counting
Extr	= Exploration Tree Creation
Con	= Enforce Consistency Constraints
Syn	= Construct Synthetic Database
X	= Number of events * Universe size (factor in millions)

Table 6.4 Execution time (in seconds) of algorithm steps for different datasets (authmgr times are approximations due to the sensitivity of the dataset)

Figure 6.20 gives a visual comparison of the total execution time. Unsurprisingly, the pure authmgr dataset took the longest. The last column in Table 6.4, namely “X”, gives a factor that seems to correspond to the execution time. This is the number of total events times the universe size of the dataset. A higher factor means longer execution time. Also, the MSNBC dataset has an “X” factor that is somewhere between the last two authmgr datasets and in Figure 6.20 it is shown that its total execution time is also between these two datasets. Thus, this factor is useful for estimating the relationship between the data and length the program will run.

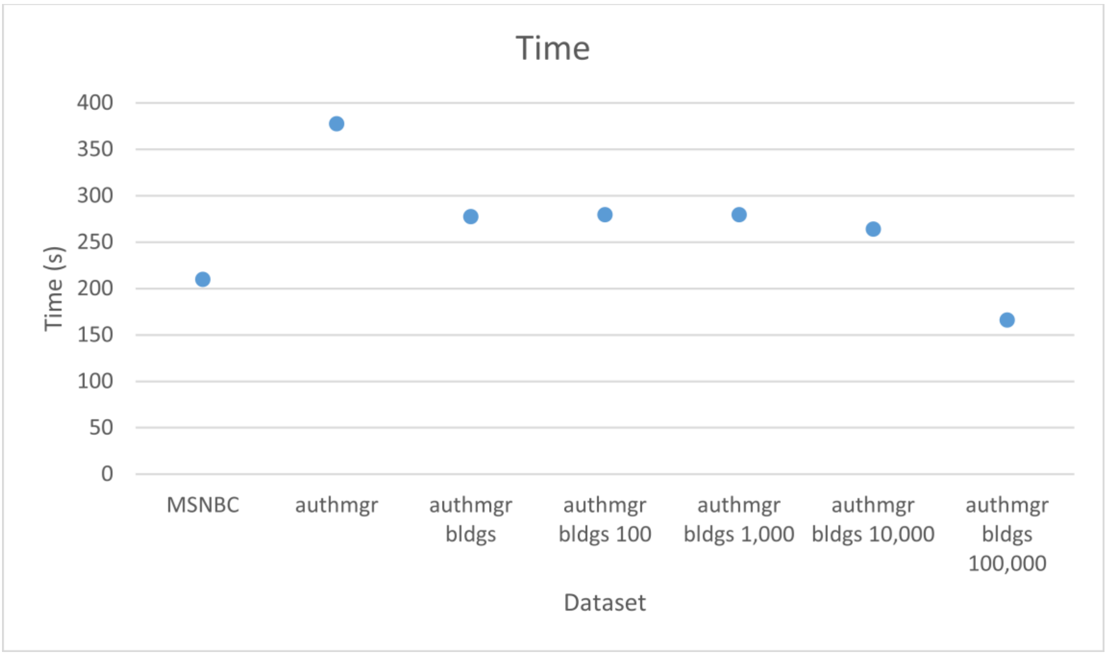


Figure 6.20 Total time to execute ngram2.0 for various datasets

This section analyzed the overall execution time of ngram2.0 without diving deep into complexity analysis. The next section will provide basis for the privacy guarantees of this algorithm.

6.5 Privacy Analysis

Ngram2.0 does not greatly change the underlying base functionality of ngrampy in terms of adding noise to count queries. Thus, the privacy analysis given by Chen in [51] using the definition of differential privacy holds for the algorithm presented in this thesis. Given that D' and D' are neighboring datasets, A is the fill exploration tree step, and $\widetilde{\mathrm{T}}'$ is the resulting noisy exploration tree, it is proven that

$$\frac{\Pr\left[\mathrm{A}(\mathrm{D}') = \widetilde{\mathrm{T}}'\right]}{\Pr\left[\mathrm{A}(\ C) = \widetilde{\mathrm{T}}'\right]} \leq \mathrm{e}^{\varepsilon}$$

This means that ngram2.0 guarantees ε-differential privacy.

The recommended parameter values should be used. Note that for a specific unique dataset, if different input parameters are chosen based on achieving better utility, then ε-differential privacy may not be achieved since the original raw data is being used to choose which synthetic output dataset to use.

There are a few techniques that have been used to prove and provide justification that implementations of differential privacy are correct. A summary of many of these techniques is given by Barthe, et al. [86]. These techniques include languages such as Fuzz and DFuzz, and relational Hoare logic. These methods would provide a full privacy analysis of the implementation, but they are non-trivial to implement. Also, the majority of them also focus on the interactive setting, as opposed to the non-interactive setting used here [87]. It is left to future work to provide such a full analysis of the implementation. For this thesis, a simpler method is used. In ngram2.0, the differential privacy guarantee is dependent on the Laplacian mechanism. At that point. This means that after the Exploration Tree Creation phase, the synthetic data is differentially private, and the following phases are post-processing phases that edit the data to raise the utility level. Thus, if the Laplacian noise mechanism is implemented correctly, differential privacy should be correctly guaranteed.

From Section 5.3.3, it is shown that the Laplacian noise is added as:

$$Y \leftarrow sign(r) * \lambda \ln(1 - 2|r|)\,, where\; r = R - 0.5$$

according to [72]. This formula is used in to [72] and many other papers on differential privacy. Isolating the function that was written to provide this calculation, and running it in a loop 1,000 times with ε = 1, $\ell_{\max}$ = 10 produces the graph in Figure 6.21. Note that an input to this formula is a random variable, so the data is sorted based on value to create this graph. It does match what the noise should look like according to the above formula which is graphed in Figure 6.22, besides the opposite signs, with the same parameters ε = 1, $\ell_{\max}$ = 10.

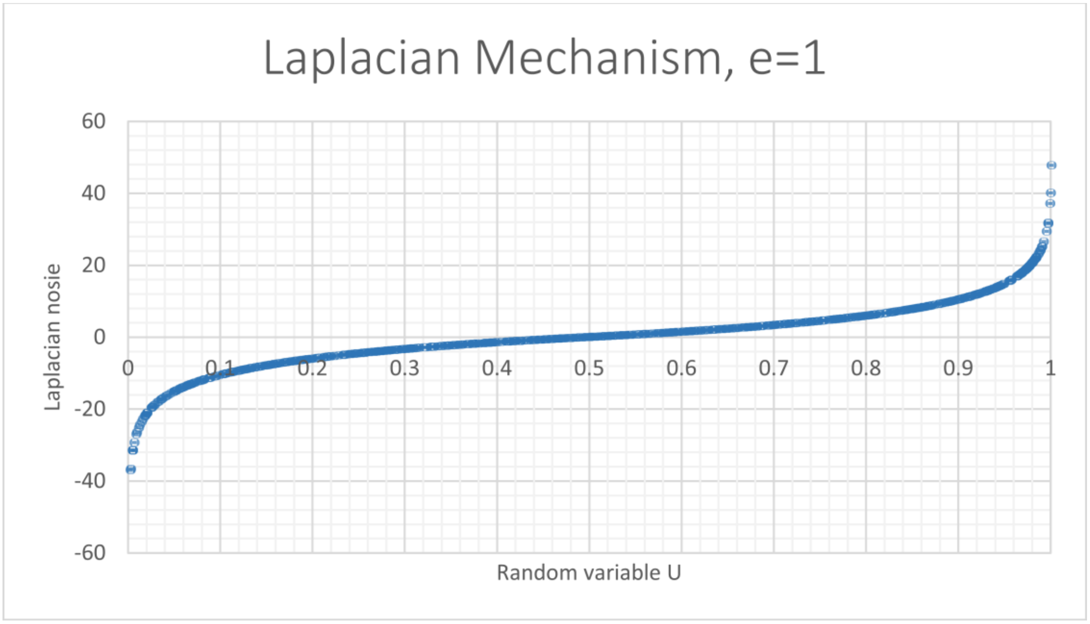


Figure 6.21 Laplace Mechanism noise output from implementation

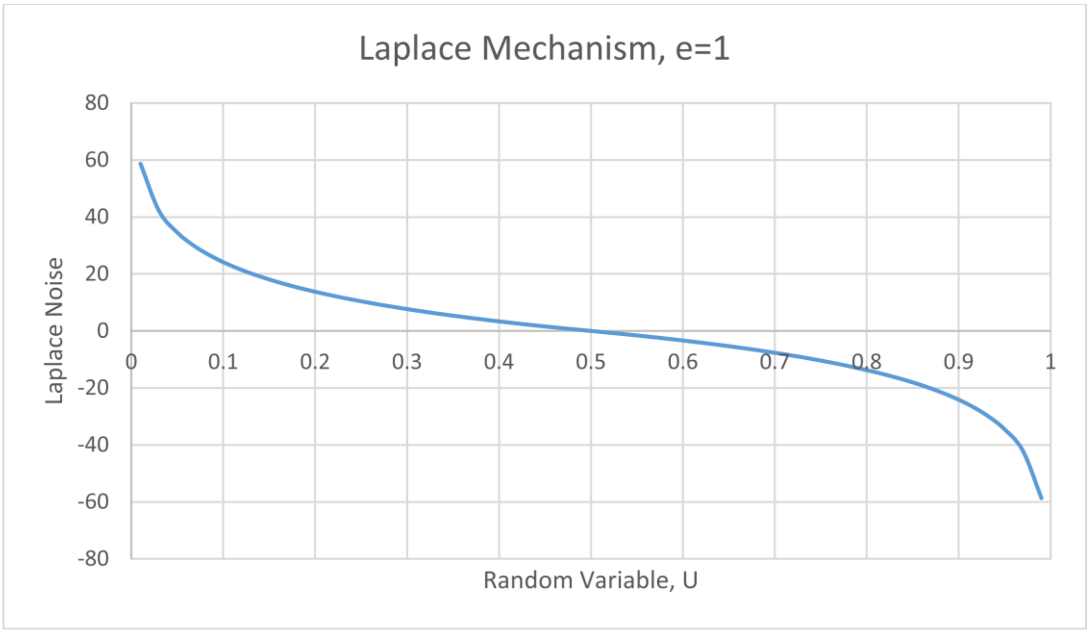


Figure 6.22 Laplace Mechanism noise output from formula

Thus, since the Laplace Mechanism implemented in ngram2.0 matches the formulaic output of what a Laplace Mechanism should output, the privacy mechanism is correctly implemented and thus ngram2.0 correctly implements differential privacy, at least at this basic level.

It is supported that ngram2.0 achieves ε-differential privacy. The last section will conclude this chapter.

6.6 Conclusions

This chapter used three utility measures to analyze ngram2.0, and then evaluated these results. It started by ensuring that ngram2.0 reached the utility level of the implementation of the algorithm it is based on, ngrampy, with a sequential dataset, MSNBC. Then it went on to analyze the impact of different parameters for ngram2.0 using various forms of the trajectory dataset, authmgr. A generalized form of the authmgr dataset, where APs are generalized to building names, resulted in much better utility than pure authmgr data. It was also verified that smaller time sensitivities result in poorer utility. A universe density calculation was introduced and concluded that the number of connections between nodes should be at least 100 times the number of universe items for adequate utility. In terms of execution time, much of that is attributed to universe creation, counting n-grams, and creating the synthetic database. It was also shown that ngram2.0 guarantees differential privacy. Overall, ngram2.0 provides an average utility result for modified authmgr data.

7 Conclusions and Future Work

This thesis has presented an algorithm and implementation thereof which provides a means for publishing authmgr and similar data. An overview of various privacy definitions, including the chosen differential privacy, and PPTDP techniques were discussed along with their advantages and shortcomings. Then pertinent algorithmic background details were discussed, namely the basis of differential privacy and the n-gram model. Next the algorithm ngram2.0 was presented. This is based off the n-gram algorithm by Chen et al. [51], but extends this to handle spatiotemporal data and changes implementation details to use C++ instead of Python to better guarantee differential privacy, make the algorithm faster, and simplify the code. Results showed that an edited authmgr datasets where APs were generalized to building names provided around the same utility results as the algorithm presented by Chen et al., which itself improved on utlitity of previous algorithms. The following gives a description of possible future work and last thoughts before concluding.

7.1 Future Work

The previous chapters described a working algorithm which provides adequate utility for authmgr log data. With the code in place, there are many paths that future work could follow to improve it.

There are a few changes that could improve the overall structure. To make ngram2.0 more available to the desired application, it should be restructured to fit within the open source Elastic stack in order to perform the data manipulation at the point of collection and storage. The framework could also be finished to allow better addition of future algorithms.

A number of improvements could be done with the validation and resulting data manipulation. More data analysis tasks could be run in addition to CQ and FSPM to analyze additional features of the dataset. These should be chosen based on analyzing the realism of the resulting dataset, studying further data algorithms and trajectory characteristic, and based on further analysis of what researchers will be using the data for. A layering of the trajectories onto a map would provide a helpful visualization for comparing the utility of the resulting synthetic dataset against the original raw dataset. This has been done in previous research, such as DPComp [88] which has an online prototype example, but there are two main hurdles. First, the network data sources used in this thesis do not have GPS coordinates. To get around this, the visualization could be a graph instead of a map where the x-axis is the building name and the y-axis is a representation of the name of the APs. Or the APs could be placed based off of their relative locations to each other. This would show a map structure that could be visually compared, but would not show what the trajectories actually look like. Second, the data is four dimensional. That is, there is an x-y-z location and a time for each AP since each AP is located on different floors of a building.

Along with the previously described improvement, GPS locations of the APs could improve the synthetic database result in other ways as described in Section 2.4. First, the choice of universe items could be reduced to those locations in close proximity to the current locations based off trajectory speed, as in He et al. [52]. This would speed up the algorithm and make the resulting trajectories more realistic. Second, this added data could be used to analyze the adherence to realistic qualities of the resulting trajectories. The one difficulty with this is that the whole campus is not covered with APs so there could be jumps in which AP is being connected to.

Future improvements could be made on the algorithm structure itself. It may be found that differential privacy is not the ideal privacy guarantee to use for this application. Another solution could be mobility models such as SMOOTH [41] where they create a mobility model that is both simple and realistic, improving on synthetic and trace-based models that focused on one or the other. In order to verify that their model adheres to real mobility patterns, they compare the results of complementary cumulative distribution functions of inter-contact times, contact durations, and contact numbers on both real data and the synthetic data created from their mobility model. In this technique, a model would be created based off the characteristics of the real dataset, such as location, velocity, and acceleration of devices in the network, and then create synthetic datasets from this model. Further details would have to be added into this model accounting for time of day, day of the week, and other temporal information. These mobility models could also be used alongside ngram2.0 to provide validation of the resulting differentially private synthetic dataset.

There are also other related concepts that should be built off of this effort. The ngram2.0 algorithm focuses on publishing one dataset, but there is a need to publish a wide range of related network datasets. The biggest hurdle with this endeavor is relational qualities between the datasets. For example, when a device authentications with a wireless network, multiple logs are created at different levels of the dataset, and all these logs are stored in different datasets. Another such improvement would be anonymzing with relation to identifier relations in the dataset. Instead of just using one identifier, a future algorithm could work with multiple identifiers. For example, in the authmgr dataset a person (identifier #1) can have multiple devices (identifier #2). So combining the approaches of anonymizing trajectory and associated metadata together would be a significant addition, as mentioned in Section 4.1.

The ngram2.0 algorithm provides an adequate anonymization scheme for authmgr data, but these features just described would greatly expand the usability of this algorithm. It is left to future researchers to construct a more robust implementation based on some of these features.

7.2 Conclusions

This thesis presented an algorithm which provides a means for publishing singular data sources from a wireless network and is useful for authmgr data with generalized location points. In doing so, it explains implementation considerations for the specific data application, namely network data, and the algorithmic components used, as discussed next. The algorithm extracts high quality n-grams from the original dataset, adds noise to these n-grams, and releases a synthetic database of trajectories. Note that this algorithm reads in row data to create sequences of spatiotemporal points, also known as trajectories. It uses the formal algorithmic privacy guarantees of differential privacy and the n-gram model. The former provides a provable avenue for discussing the privacy guarantees of an algorithm and in this case is implemented through addition of noise to the dataset. The latter provides a means for retaining and using sequential data to reconstruct a synthetic database. The use of differential privacy ensures user private data is not revealed. Since the data is assumed to be independent, these guarantees hold. There were also implementation features that protect against differential privacy attacks, namely using fixed point numbers, instead of floating point, in the noise calculation. This algorithm also provides high utility for data with a large number of events compared to the universe size. This was from the comparison of this high dimensional data with CQ and FSPM results from both the original raw and synthetic datasets, and comparison of these results against previous PPTDP algorithms.

Improvements could be made on the algorithm execution speed, and further research is needed to improve utility results for datasets with higher data sparsity. The code base can be used directly by data holders to publish their data, and should provide an easy framework for editing the algorithm implemented.

Overall, ngram2.0 adds to the literature for Privacy Preserving Trajectory Data Publication, providing an algorithm to publish sequential spatiotemporal network data. It starts the process towards the ability to publish network data from large ISPs, such as Virginia Tech, to researchers for the purpose of movement trends, user density counts, and a plethora of further research topics.

REFERENCES

[1] A. Balachandran, G. M. Voelker, P. Bahl, and P. V. Rangan, “Characterizing User Behavior and Network Performance in a Public Wireless LAN,” in *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, New York, NY, USA, 2002, pp. 195–205.

[2] D. Kotz and K. Essien, “Analysis of a Campus-wide Wireless Network,” in *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, New York, NY, USA, 2002, pp. 107–118.

[3] P. S. Prasad and P. Agrawal, “Movement prediction in wireless networks using mobility traces,” in *2010 7th IEEE Consumer Communications and Networking Conference*, 2010, pp. 1–5.

[4] U. Kumar, J. Kim, and A. Helmy, “Comparing Wireless Network Usage: Laptop vs Smart-phones,” in *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, New York, NY, USA, 2013, pp. 243–246.

[5] M. M. Mulhanga, S. R. Lima, and P. Carvalho, “Characterising university wlans within eduroam context,” in *Smart spaces and next generation wired/wireless networking*, Springer, 2011, pp. 382–394.

[6] T. Henderson, D. Kotz, and I. Abyzov, “The Changing Usage of a Mature Campus-wide Wireless Network,” in *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, New York, NY, USA, 2004, pp. 187–201.

[7] J. Steshenko, V. G. Chaganti, and J. Kurose, “Mobility in a Large-scale WiFi Network: From Syslog Events to Mobile User Sessions,” in *Proceedings of the 17th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, New York, NY, USA, 2014, pp. 331–334.

[8] W. Weng, K. Lei, K. Xu, X. Liu, and T. Sun, “Internet Traffic Analysis in a Large University Town: A Graphical and Clustering Approach,” in *Web-Age Information Management*, 2016, pp. 378–389.

[9] C. Dwork, “The Promise of Differential Privacy. A Tutorial on Algorithmic Techniques.,” *Microsoft Res.*, Oct. 2011.

[10] G. Simson, “De-identifiying Government Datasets (Draft 2),” National Institute of Standards and Technology (NIST), NIST Special Publication 800–188 DRAFT2, Dec. 2016.

[11] L. Sweeney, “k-anonymity: a model for protecting privacy,” *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 557–570, 2002.

[12] Michael Barbaro and Tom Zeller Jr., “A Face Is Exposed for AOL Searcher No. 4417749,” *New York Times*, p. A1, 09-Aug-2006.

[13] Arvind Narayanan and Vitaly Shmatikov, “Robust De-anonymization of Large Sparse Datasets,” presented at the Proc. 29th IEEE Symposium on Security and Privacy, 2008.

[14] Y.-A. de Montjoye, C. A. Hidalgo, M. Verleysen, and V. D. Blondel, “Unique in the Crowd: The privacy bounds of human mobility,” *Sci. Rep.*, vol. 3, p. 1376, Mar. 2013.

[15] K. Tan, G. Yan, J. Yeo, and D. Kotz, “Privacy analysis of user association logs in a large-scale wireless LAN,” in *2011 Proceedings IEEE INFOCOM*, 2011, pp. 31–35.
[16] T. Higuchi, H. Yamaguchi, and T. Higashino, “Trajectory identification based on spatio-temporal proximity patterns between mobile phones,” *Wirel. Netw.*, vol. 22, no. 2, pp. 563–577, Feb. 2016.
[17] C. Dwork, “Differential Privacy: A Survey of Results,” in *Theory and Applications of Models of Computation*, 2008, pp. 1–19.
[18] N. R. Council, *Private Lives and Public Policies: Confidentiality and Accessibility of Government Statistics*. 1969.
[19] 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*, New York, NY, USA, 2012, pp. 213–221.
[20] P. Samarati and L. Sweeney, “Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression,” Computer Science Laboratory, SRI International, Technical Report SRI-CSL-98-04, 1998.
[21] N. Li, T. Li, and S. Venkatasubramanian, “t-Closeness: Privacy Beyond k-Anonymity and l-Diversity,” in *2007 IEEE 23rd International Conference on Data Engineering*, 2007, pp. 106–115.
[22] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, “L-diversity: Privacy Beyond K-anonymity,” *ACM Trans Knowl Discov Data*, vol. 1, no. 1, Mar. 2007.
[23] H. Wang, J. Han, J. Wang, and L. Wang, “(k, ε)-Anonymity: An anonymity model for thwarting similarity attack,” in *2013 IEEE International Conference on Granular Computing (GrC)*, 2013, pp. 332–337.
[24] Q. Wei, Y. Lu, and L. Zou, “ε-inclusion: privacy preserving re-publication of dynamic datasets,” *J. Zhejiang Univ.-Sci. A*, vol. 9, no. 8, pp. 1124–1133, Aug. 2008.
[25] 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*, New York, NY, USA, 2008, pp. 265–273.
[26] D. Kifer, “Attacks on Privacy and deFinetti’s Theorem,” in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2009, pp. 127–138.
[27] R. C.-W. Wong, A. W.-C. Fu, K. Wang, P. S. Yu, and J. Pei, “Can the Utility of Anonymized Data Be Used for Privacy Breaches?,” *ACM Trans Knowl Discov Data*, vol. 5, no. 3, pp. 16:1–16:24, Aug. 2011.
[28] C. Dwork, “Differential Privacy,” in *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II*, Berlin, Heidelberg, 2006, pp. 1–12.
[29] C. Dwork and A. Roth, “The Algorithmic Foundations of Differential Privacy,” *Found Trends Theor Comput Sci*, vol. 9, no. 3–4, pp. 211–407, Aug. 2014.
[30] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, “Our data, ourselves: Privacy via distributed noise generation,” in *Annual International*

Conference on the Theory and Applications of Cryptographic Techniques, 2006, pp. 486–503.
[31] Z. Teng and W. Du, “Comparisons of K-Anonymization and Randomization Schemes under Linking Attacks,” in *Sixth International Conference on Data Mining (ICDM’06)*, 2006, pp. 1091–1096.
[32] “OnTheMap.” [Online]. Available: https://onthemap.ces.census.gov/. [Accessed: 06-Jul-2017].
[33] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber, “Privacy: Theory meets Practice on the Map,” in *2008 IEEE 24th International Conference on Data Engineering*, 2008, pp. 277–286.
[34] Ú. Erlingsson, V. Pihur, and A. Korolova, “RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response,” 2014, pp. 1054–1067.
[35] “Apple previews iOS 10, biggest iOS release ever,” *Apple Newsroom*, 13-Jun-2016. [Online]. Available: https://www.apple.com/newsroom/2016/06/apple-previews-ios-10-biggest-ios-release-ever/. [Accessed: 06-Jul-2017].
[36] J. Bambauer, K. Muralidhar, and R. Sarathy, “Fool’s Gold: An Illustrated Critique of Differential Privacy,” *Vanderbilit J. Entertain. Technol. Law*, vol. 16, no. 4, pp. 701–752, 2014.
[37] F. D. McSherry, “Differential privacy for dummies,” 03-Feb-2016. .
[38] J. Bambauer and K. Muralidhar, “Diffensive Privacy,” *INFO/LAW*, 17-May-2016. .
[39] C. Dwork and G. N. Rothblum, “Concentrated Differential Privacy,” *ArXiv160301887 Cs*, Mar. 2016.
[40] Rathindra Sarathy and Krish Muralidhar, “DifferePrivacy for Numeric Data,” presented at the United Nations Statistical Commisiononomic Commision for Europe Conference of European Statisticians, Bilbao, Spain, 2009.
[41] K. Nissim, S. Raskhodnikova, and A. Smith, “Smooth Sensitivity and Sampling in Private Data Analysis,” in *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, New York, NY, USA, 2007, pp. 75–84.
[42] R. Sarathy and K. Muralidhar, “Some Additional Insights on Applying Differential Privacy for Numeric Data,” in *SpringerLink*, 2010, pp. 210–219.
[43] R. Sarathy and K. Muralidhar, “Evaluating Laplace Noise Addition to Satisfy Differential Privacy for Numeric Data,” *Trans Data Priv.*, vol. 4, no. 1, pp. 1–17, Apr. 2011.
[44] D. Kifer and A. Machanavajjhala, “No free lunch in data privacy,” in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011, pp. 193–204.
[45] A. Haeberlen, B. C. Pierce, and A. Narayan, “Differential Privacy Under Fire,” in *Proceedings of the 20th USENIX Conference on Security*, Berkeley, CA, USA, 2011, pp. 33–33.
[46] J. Hsu *et al.*, “Differential Privacy: An Economic Method for Choosing Epsilon,” in *2014 IEEE 27th Computer Security Foundations Symposium*, 2014, pp. 398–410.
[47] R. Chen, B. Fung, and B. C. Desai, “Differentially private trajectory data publication,” *ArXiv Prepr. ArXiv11122020*, 2011.
[48] C. Dwork, F. McSherry, K. Nissim, and A. Smith, “Calibrating Noise to Sensitivity in Private Data Analysis,” in *Theory of Cryptography*, 2006, pp. 265–284.

[49] 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*, New York, NY, USA, 2013, pp. 12:1–12:12.

[50] D. Shao, K. Jiang, T. Kister, S. Bressan, and K.-L. Tan, “Publishing Trajectory with Differential Privacy: A Priori vs. A Posteriori Sampling Mechanisms,” in *Database and Expert Systems Applications*, 2013, pp. 357–365.

[51] R. Chen, G. Acs, and C. Castelluccia, “Differentially Private Sequential Data Publication via Variable-length N-grams,” in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, New York, NY, USA, 2012, pp. 638–649.

[52] X. He, G. Cormode, A. Machanavajjhala, C. M. Procopiuc, and D. Srivastava, “DPT: differentially private trajectory synthesis using hierarchical reference systems,” *Proc. VLDB Endow.*, vol. 8, no. 11, pp. 1154–1165, 2015.

[53] F. McSherry and K. Talwar, “Mechanism Design via Differential Privacy,” *Microsoft Res.*, Oct. 2007.

[54] J. Hua, Y. Gao, and S. Zhong, “Differentially private publication of general time-serial trajectory data,” in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 549–557.

[55] M. Li, L. Zhu, Z. Zhang, and R. Xu, “Differentially Private Publication Scheme for Trajectory Data,” in *2016 IEEE First International Conference on Data Science in Cyberspace (DSC)*, 2016, pp. 596–601.

[56] F. Prasser, “ARX,” 01-May-2013. [Online]. Available: http://arx.deidentifier.org/. [Accessed: 08-Apr-2017].

[57] F. Prasser and F. Kohlmayer, “Putting Statistical Disclosure Control into Practice: The ARX Data Anonymization Tool,” in *Medical Data Privacy Handbook*, A. Gkoulalas-Divanis and G. Loukides, Eds. Springer International Publishing, 2015, pp. 111–148.

[58] M. Kantarcioglu, “UTD Anonymization ToolBox,” 29-Mar-2010. [Online]. Available: http://www.cs.utdallas.edu/dspl/cgi-bin/toolbox/index.php?go=home. [Accessed: 08-Apr-2017].

[59] wanghaisheng, *Cornell-Anonymization-Toolkit: mirror of Cornell Anonymization Toolkit*. 2016.

[60] X. Xiao, G. Wang, and J. Gehrke, “Interactive Anonymization of Sensitive Data,” in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2009, pp. 1051–1054.

[61] C. Dai, G. Ghinita, E. Bertino, J.-W. Byun, and N. Li, “TIAMAT: A Tool for Interactive Analysis of Microdata Anonymization Techniques,” *Proc VLDB Endow*, vol. 2, no. 2, pp. 1618–1621, Aug. 2009.

[62] “The SECRETA system.” [Online]. Available: http://users.uop.gr/~poulis/SECRETA/index.html. [Accessed: 07-Jun-2017].

[63] G. Poulis, A. Gkoulalas-Divanis, G. Loukides, S. Skiadopoulos, and C. Tryfonopoulos, “SECRETA: A Tool for Anonymizing Relational, Transaction and RT-Datasets,” pp. 83–109, 2015.

[64] “Open Anonymizer download | SourceForge.net.” [Online]. Available: https://sourceforge.net/projects/openanonymizer/. [Accessed: 07-Jun-2017].

[65] Konrad Stark, “Scientific Workflows, Data Provenance Management and Data Anonymization in Context of the Genome Austria Tissue Bank,” Universität Wien, 2013.
[66] Johannes Drepper, “V086-01 Anon-Tool,” 20. [Online]. Available: http://www.tmf-ev.de/Themen/Projekte/V08601_AnonTool.aspx. [Accessed: 07-Jun-2017].
[67] P.-P. de Wolf, “mu-ARGUS (μ-ARGUS),” *Statistical Disclosure Control*, 20-Dec-2012. [Online]. Available: http://neon.vb.cbs.nl/casc/mu.htm. [Accessed: 18-Feb-2017].
[68] M. Templ, A. Kowarik, and B. Meindl, *sdcMicro: Statistical Disclosure Control Methods for Anonymization of Microdata and Risk Estimation*. 2017.
[69] “Privacy Analytics: Risk-based De-identification Software and Services for HIPAA Compliance,” *Privacy Analytics*. [Online]. Available: https://privacy-analytics.com/. [Accessed: 07-Jun-2017].
[70] X. He, N. Raval, and A. Machanavajjhala, “A Demonstration of VisDPT: Visual Exploration of Differentially Private Trajectories,” *Proc VLDB Endow*, vol. 9, no. 13, pp. 1489–1492, Sep. 2016.
[71] C. Dwork, “A Firm Foundation for Private Data Analysis,” *Commun ACM*, vol. 54, no. 1, pp. 86–95, Jan. 2011.
[72] I. Mironov, “On Significance of the Least Significant Bits for Differential Privacy,” in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, New York, NY, USA, 2012, pp. 650–661.
[73] N. Li, W. Qardaji, and D. Su, “On Sampling, Anonymization, and Differential Privacy: Or, k-Anonymization Meets Differential Privacy,” *ArXiv11012604 Cs*, Jan. 2011.
[74] F. McSherry, “Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis,” *Commun ACM*, vol. 53, no. 9, pp. 89–97, Sep. 2010.
[75] M. Hay, V. Rastogi, G. Miklau, and D. Suciu, “Boosting the Accuracy of Differentially Private Histograms Through Consistency,” *Proc VLDB Endow*, vol. 3, no. 1–2, pp. 1021–1032, Sep. 2010.
[76] A. Narayan, A. Feldman, A. Papadimitriou, and A. Haeberlen, “Verifiable Differential Privacy,” in *Proceedings of the Tenth European Conference on Computer Systems*, New York, NY, USA, 2015, pp. 28:1–28:14.
[77] John McFarlane, “Composition of Arithmetic Types (Document number: P0554R0),” 06-Feb-2017. [Online]. Available: http://open-std.org/JTC1/SC22/WG21/docs/papers/2017/p0554r0.html. [Accessed: 24-Feb-2017].
[78] R. J. Jenkins Jr., “ISAAC,” in *Proceedings of the Third International Workshop on Fast Software Encryption*, London, UK, UK, 1996, pp. 41–49.
[79] J. Zhang, X. Xiao, and X. Xie, “PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions,” in *Proceedings of the 2016 International Conference on Management of Data*, New York, NY, USA, 2016, pp. 155–170.
[80] X. Xiao, G. Bender, M. Hay, and J. Gehrke, “iReduct: Differential Privacy with Reduced Relative Errors,” in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2011, pp. 229–240.

[81] X. Xiao, G. Wang, and J. Gehrke, “Differential Privacy via Wavelet Transforms,” *IEEE Trans Knowl Data Eng*, vol. 23, no. 8, pp. 1200–1214, Aug. 2011.

[82] D. Burdick, M. Calimlim, and J. Gehrke, “MAFIA: a maximal frequent itemset algorithm for transactional databases,” in *Proceedings 17th International Conference on Data Engineering*, 2001, pp. 443–452.

[83] “MAFIA: Maximal Frequent Itemsets.” [Online]. Available: http://himalaya-tools.sourceforge.net/Mafia/. [Accessed: 02-Sep-2017].

[84] “UCI Machine Learning Repository: MSNBC.com Anonymous Web Data Data Set.” [Online]. Available: http://archive.ics.uci.edu/ml/datasets/msnbc.com+anonymous+web+data. [Accessed: 15-Aug-2017].

[85] F. McSherry and R. Mahajan, “Differentially-private Network Trace Analysis,” in *Proceedings of the ACM SIGCOMM 2010 Conference*, New York, NY, USA, 2010, pp. 123–134.

[86] G. Barthe, M. Gaboardi, J. Hsu, and B. Pierce, “Programming Language Techniques for Differential Privacy,” *ACM SIGLOG News*, vol. 3, no. 1, pp. 34–53, Feb. 2016.

[87] A. Narayanan, “Data privacy : the non-interactive setting,” May 2009.

[88] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, D. Zhang, and G. Bissias, “Exploring Privacy-Accuracy Tradeoffs Using DPComp,” in *Proceedings of the 2016 International Conference on Management of Data*, New York, NY, USA, 2016, pp. 2101–2104.

Appendix A: Software Usage

This appendix is an overview of the software code and how to use it. For more information, please see the README.md file in the repository.

A.1 Installation

In order to run some of the software, a linux platform is required. This can either be a Windows machine with Cygwin (or another unix-style command line program for Windows) or a Linux machine. The main development for this project was done on a Windows 8.1 machine running Cygwin 2.9.0. Though the main testing environments were Linux machines.

The first step to use this software is to clone the git repository:

```
    git clone https://code.vt.edu/LAARG/differentialprivacy
```

Or previously:

```
    git clone https://git.cirt.vt.edu/LAARG/differentialprivacy
```

Once the repository is cloned, follow the instructions in the README.md files to run the program. All outside libraries are currently in the repository and this file contains notes on any edits done on the outsides sources used.

A.2 Usage

In order to get the data out of the ELK stack, es2csv is used. From a computer with access to the ELK stack (ie Peach in the ITSL), run the following command

```
   es2csv -u (ip of ELK):(port) -i logstash-2016.11.02 -q
   'syslog_program:authmgr' -o radiusd_2016_11_03.csv
```

This will produce a CSV file from the logstash authentication manager file.

To comply with privacy regulations, this file was then put through a simple de-identification program which hashed the identifiers with a salt. Ngram2.0 does do this as well, so with regular usage this step would not have to be taken, but for the purposes of testing, this step was taken.

The next step is to run this CSV file through ngram2.0. Again the README file in the repository contains a more complete description of the commands, but the following is a simple command that can be run on the raw data file

```
   ./trajanon.out -f authmgr.csv -a TrajAnon
```

where `authmgr.csv` is the filename from the previous step.

This will output intermediary filed, the final anonymized dataset, and results of the CQ validation code. The FSPM validation must be run separately at this point.

The code can also be used with an exposed API. Though future work will have to be done to make sure all the API commands function as expected for other algorithms implemented.