Distribution Agreement

In presenting this thesis or dissertation as a partial fulfillment of the requirements for an advanced degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis or dissertation in whole or in part in all forms of media, now or hereafter known, including display on the world wide web. I understand that I may select some access restrictions as part of the online submission of this thesis or dissertation. I retain all ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Signature:

_____ _____

Luca Bonomi Date

Big Data goes Personal: Privacy and Social Challenges

By

Luca Bonomi
Doctor of Philosophy

Computer Science and Informatics

Li Xiong, Ph.D.
Advisor

Michelangelo Grigni, Ph.D.
Committee Member

James J. Lu, Ph.D.
Committee Member

Accepted:

Lisa A. Tedesco, Ph.D.
Dean of the Graduate School

Date

Big Data goes Personal: Privacy and Social Challenges

By

Luca Bonomi
MS, Computer Engineering, 2008

Advisor: Li Xiong, Ph.D.

An abstract of
A dissertation submitted to the Faculty of the Graduate School
of Emory University in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Informatics
2015

Abstract

Big Data goes Personal: Privacy and Social Challenges
By Luca Bonomi

The Big Data phenomenon is posing new challenges in our modern society. In addition to requiring information systems to effectively manage high-dimensional and complex data, the privacy and social implications associated with the data collection, data analytics, and service requirements create new important research problems. First, the high volume of personal data generated by users' devices (e.g. credit card transactions, GPS trajectories from mobile devices and medical data) can be used, much like a fingerprint, to identify the person who created it with the risk of disclosing sensitive information such as: political inclination, financial status and medical condition. Second, popular social networks (e.g. Facebook, Foursquare, Yelp) not only enable users to share locations and preference but also create opportunities for them to establish complex interactions (e.g. forming communities, planning trip). This creates the needs for location based services to provide services to groups of users rather than individuals. In this dissertation, we present effective solutions for both these privacy and social challenges. In the privacy domain, we propose new privacy preserving techniques to provide individual users with formal guarantee of privacy while at the same time preserve meaningful information of the data released. We demonstrate the effectiveness of our solutions in different domains such as: sequential pattern mining, record linkage, and computation of statistics over data streams. In the social domain, we propose a new type of group query aiming to find a route that all users can traverse while maximizing the group preference for the locations jointly visited. The ability of solving such query can greatly benefit many existing and emerging tools that allow users to share route information (e.g. Uber, Waze) and plan group outings or trips (e.g. QuickCliqs). Extensive empirical studies demonstrate the effectiveness of our solutions and provide us with important insights for future research directions.

Big Data goes Personal: Privacy and Social Challenges

By

Luca Bonomi
MS, Computer Engineering, 2008

Advisor: Li Xiong, Ph.D.

A dissertation submitted to the Faculty of the Graduate School
of Emory University in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Informatics
2015

Acknowledgments

First, I would like to express my deepest gratitude to my advisor, Dr. Li Xiong, for her continue support throughout my dissertation. Her guidance, caring, and patience helped me during my research. I would like to thank my committee members, Dr. Michelangelo Grigni and Dr. James J. Lu, for their precious advices and insights during the course of my research. Their time and effort are highly appreciated. My gratitude goes also to Dr. Benjamin C. M. Fung, Dr. Vaidy Sunderam, and Dr. Shun Yan Cheung for their valuable suggestions and comments. I also would like to thank the Math&CS staff for their help during all these years.

In memory of A.A. 1948-2015.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The recent and rapid growth of data produced in our society is posing important challenges in database system design. In addition to requiring efficient information systems to manage high-dimensional and complex data generated by multiple devices, the large amount of data is creating new privacy and social concerns.

The first part of this dissertation aims to address the privacy challenges that arise in the data collection and data management (e.g. sharing and analytics) associated with the big data phenomenon. In fact, as the data volume collected is tremendously increasing over the past decades, the numerous user's personal devices (e.g. smartphone, smartwatch, wristband) are creating a large portion of data that is user specific. According to the tech report in [69], the typical American office worker produces 1.8 million megabytes of data each year (5,000 megabytes/day). User's generated data is typically collected by third party with the intention of improving user's experience, providing more personalized services and perform secondary analysis. For example, in the location-based service setting, mobile devices collect user's GPS traces to provide location-based recommendation, or in the health-care domain, the collection of user's medical data aims to provide

better treatments for patients and early diagnoses. While these data analytics are beneficial for both user and external parties, they pose serious **privacy risks** for disclosing user sensitive information. In fact, this data can be used, much like a fingerprint, to identify the person who created it: your choice of movies on Netflix, the location signals emitted by your cell phone, even your buying habits recorded by credit card transactions. In fact, according to recent studies, the location of the 87% of the U.S. adult population is know via their mobile devices data. As this personal data is constantly growing, more informative the data gets creating higher risks in disclosing user's sensitive information. In the first part of this work, we address the important problem of protecting user's sensitive information. Specifically, we propose a series of new techniques base on formal statistical tools, optimization algorithms and specific domain knowledge to provide users with formal guarantee of privacy while at the same time preserve meaningful information in the data.

The second part of this work investigates the complex user-to-user iterations in the big data era and aims to define new type of database queries and services for group of users. The technological revolution that is occurring in our society and the richness of data are changing the way that users interact. Through social network, individual users find their place in communities, establish complex social relationships and participate in social events. This situation is posing new challenges in modern database systems that are facing the need of queries able to satisfy users **social needs**. Many emerging and popular tools (e.g. Uber,Waze, QuickClips) enable users to share information about routes and plan trips together allowing users to express their social needs. However, existing research are mostly limited to route queries involving single user or just beginning to look at simple group queries minimizing the distance traveled by a group of users. In this work, we make a step toward the design of location-based systems that support queries for a group of users. In particular, we propose a new type of group query which goal is to find a route that all users can traverse while maximizing the group pref-

erence for the locations jointly visited. We believe that our route problem could greatly improve existing group routing or trip planning applications and our formulation balances the users social needs as well as their individual preferences.

1.1.1 Privacy Needs

A first component of this work [14, 15, 10, 12, 9] is centered around the **user's privacy needs** and it focuses on the design of efficient privacy preserving solutions for data analytics. Sharing user's data poses serious privacy concerns, as such data contain behavioral patterns that may disclose personal sensitive information when data are mined by third party. As in the recent Target incident, department stores may look at individual purchase history to predict whether the customer is changing buying behavior and it may disclose that the costumer is going through a major life event, e.g. pregnancy. Therefore, it is crucial to design privacy preserving techniques that **protect user's sensitive information** while at the same time enable applications to obtain meaningful mining results.

The current state-of-the-art paradigm for privacy-preserving data publishing is based on the notion of **differential privacy** [29]. Such privacy model requires that the aggregate statistics reported by a data publisher be perturbed by a randomized algorithm $\mathcal{A}$ prior the publication, so that the output of $\mathcal{A}$ remains roughly the same even if any single user's record in the input data is arbitrarily modified. Therefore, by observing the output of $\mathcal{A}$, an adversary will not be able to infer much about any single data record in the input, and thus privacy is protected. In the last decade, a plethora of works have been developed for differentially private data publishing [30, 56, 36] providing privacy solutions in a variety of settings.

In this work, we consider the privacy risks in releasing data statistics in two different scenarios. First, we consider an off-line setting, where the data is given in input and the statistics are published once. Specifically, we consider the problem of releasing the top frequent patterns from aggregated user sequential data while

providing user-privacy level. These patterns have the important property of preserving the temporal relationship between events and for this reason are extremely popular in several areas, such as: computational biology, location-based service and web-browsing monitoring. The design of privacy preserving solutions for sequential pattern mining presents two major challenges: (1) *high-dimensional* output space (i.e. high number of patterns), and (2) *high-sensitivity* query (i.e. privacy has a great impact on the final utility). Second, we consider an on-line setting, where the input is a data stream and the released statistics have to be updated each time a new data element arrives in the stream. In this case, we analyze the problem of privately computing statistics over the data stream. The real-time requirement and long temporal observations of the data pose important challenges from the privacy and computational perspective.

1.1.2 Social Needs

A second component of this work [11] aims to address the social needs that are emerging in our society. The richness of data and users interactions in modern applications (e.g. social networks, location-based services) create opportunities for users to have a new social role. The increasing popularity of social networks, such as Facebook, Google+ and Foursquare, enriched the user to user interaction creating new opportunities and challenges in the spatial databases field. Social networks enable users to share their locations as well as their preferences, such as restaurants, museums etc. Recent works [78, 72, 58, 67] incorporate user's social information and road-network conditions (e.g. traffic) into the route recommendation process. For example, the work in [78] exploits such social information by integrating the user's profile similarity to provide a better route recommendation. Despite the use of the social information in determining user's itineraries, the query task is still targeting a single user. A new problem that arises in such a setting, is to determine a route that a group of users can share. Hashem et al. [42]

address the problem of finding a preferred route for multiple users such that the overall distance traveled by the group is minimized. Despite the effort in modeling the users' interests, this solution does not offer the flexibility needed when users have different preferences or mobility constraints.

In this work, we propose a novel query, named *Optimal Group Route* (OGR) query, which enables users to jointly plan an itinerary (i.e. a route) within their mobility constraints such that the overall preference on the locations visited by the group is maximized. The need for this type of queries naturally arise when a group of users wish to plan a group trip or a group outing together, each with certain mobility constraints and preferences on places to visit. Solving this type of query presents new challenges compared to traditional location-based query. In fact, this query aims to satisfy the group's needs, hence the computed solution has to combine user's personal preference and mobility constraints to find the optimal itinerary that all the users can jointly visit. Furthermore, we are able to show that this problem is computationally hard.

1.2 Research Contributions

The main contributions of this work consist in addressing the privacy and social aspects in designing modern database systems. First, our contributions in privacy focus on the design of algorithmic solutions to support sequential pattern mining [9, 10, 14] and data stream statistics [12]. This line of work provides an effective way to privately achieve data sharing and data re-use. These tasks are extremely beneficial in many domains, for example in health-care setting, our solutions will enable institutions to share data to advance medical research leading to cost-effective treatment plans, and better diagnose. Second, with our work on the users social aspects [11], we push the design of location-based services to a new level, where the service required comes from a group of users rather than a single individual. Our investigation on users social needs provides a better under-

standing of user-to-user interaction in modern databases systems and has potential impact on many emerging applications that consider group of users and our society. For example, our solutions can be applied in deciding traveling itinerary for group of users (e.g. commuting users) which could effectively reduce pollution and traffic in major cities.

In the rest of the section, we briefly summarize the specific contributions for each chapter of the thesis.

1.2.1 Sequential pattern mining (Chapter 3)

In this chapter, we study the problem of privately mining frequent sequential patterns from aggregated user data sequences. A first approach for differentially private mining of sequential patterns has been proposed in [23]. This solution partitions the input dataset by exploiting the string prefixes. Due to its nature, this approach is quite effective for prefix patterns but the results for substring patterns are quite poor. Recently, Chen et al. [22] proposed an alternative way for mining sequential patterns. They first reduce the dimensionality of the pattern space by restricting the mining on short patterns (n-grams), and second use the Markov assumption to construct a sanitized dataset using the noisy patterns. This approach works well for mining frequent patterns in the form of substrings, however its utility for prefixes is poor. The main reason is that the n-gram model used to construct the sanitized dataset does not consider the position of the patterns nor makes distinction between substring and prefix patterns. This results in the impossibility to recover the prefixes in the released data.

As our contributions, we propose a novel approach to effectively mine the top-k substring patterns without losing information about the frequent prefixes. We observe that mining the substring patterns directly from the data incurs a large perturbation noise due to the presence of long strings. Despite this negative result, in real datasets the majority of the strings are short and only few of them are very

long. Therefore, it is reasonable to think that a considerable number of occurrences of the frequent patterns are captured by the short strings. This observation motivates us to propose a two-phase approach [10]. In the first phase, a Markov Model is employed to privately learn the structure of the patterns with the goal of reducing the impact of the privacy on the final result. In this way, the algorithm obtains an approximate set of candidate frequent patterns which are used in the second phase to construct a sketch of the original data. In this new compact representation, the privacy impact on the utility is considerably reduced providing accurate results for the mined patterns. In the experiments, the proposed solution improves the state-of-the-art privacy preserving mining technique by enabling an accurate computation of both substring and prefix patterns.

1.2.2 Record Linkage (Chapter 4)

Record linkage is the process of identifying records that refer to the same real world entity across different sources. It is extensively used in many applications, for example, in linking medical data of the same patient across different hospitals in the country or in collecting the credit history of users from several sources. However, many of these data may contain sensitive personal information that could disclose individual privacy. In this chapter, we propose a novel secure data transformation method for linking string records in a three-parties setting.

For secure transformation techniques, the embedding of the original data into a new space usually requires the definition of common transformation criteria between the parties involved in the linkage process. The current state-of-the-art [64] uses a random base of strings to perform the embedding. This set is generated from a pool of random strings, one of the party applies some heuristic techniques to refine the strings in the base such that the error in representing the records is minimized. Therefore, in a presence of an adversary this set may leak sensitive information about the individual records when shared among the parties. In our

approach [14, 13], we employ an embedding function that is defined over a set of substrings called *base* to transform the original string records of the data holders into vectors that are matched by a third party (not necessary trusted). This set is data dependent and it preserves the privacy of the individual records in the original data. In this way, the data holders employ a map that captures the structure of their data without incurring the risk of disclosing sensitive information. We show that this transformation in a new space has several beneficial properties. First, the map preserves the structure of the original data leading to high utility results in the matching phase. Second, it does leak only a limited amount of information according to our adversary model. Finally, it allows to efficiently match records using the Euclidean distance in the embedding space which is considerably less computationally intense than the Edit distance in the original space.

We study the security and geometrical properties of our proposed technique and we demonstrate that our strategy produces optimal matching results without requiring any a priori knowledge in the embedded space. Furthermore, we present an extensive set of experiments showing that our approach obtains comparable utility results with the state of the art secure transformation for record linkage proposed in [64], while providing rigorous privacy guarantees and better scalability. Finally, we implement our technique and propose a LinkIT framework [15] for privacy preserving record linkage.

1.2.3 Data anlytics over streams (Chapter 5)

In this chapter, we address the problem of privately computing statistics over data streams. As opposed to **offline methods**, our techniques enable a continues and private release of the required data statistics in an **online fashion**.

The task that we consider consists in designing solutions for efficiently computing of the length of the longest increasing subsequence (LIS) over individual user stream data. The computation of the LIS provides useful information about

the sortedness of the data stream and it can be used to detect trends in time-series data. Furthermore, this problem rises new challenges compared to the traditional privacy setting. First of all, privacy requirements in protecting sensitive information for this ordered statistic have a greater impact on the final utility. Count based statistic over a stream are typically computed by decomposition which leads to a considerable reduction of perturbation noise required by the privacy mechanism. However, ordered statistics are generally not easy to approximate via decomposition since they require a global view of the entire stream. Second, the LIS has higher memory requirements compared to standard counting based statistics (e.g. counts, heavy hitters). In fact, it has been shown in [38] that there exists a space lower bound of $\Omega(T)$ for any randomized algorithm that computes the LIS exactly over a stream of length T. This strong separation between count based functions and LIS impacts both efficiency and utility of the solutions for this problem.

To address these challenges, we propose a series of solutions for privately computing the LIS while minimizing the error introduced by the perturbation and approximation [12]. We propose a decomposition framework for approximating the length of the LIS using local information in the stream. First, we formally analyze the performance of this solution and we demonstrate how this technique allows us to reduce the error due to perturbation noise from the privacy mechanism. Second, we propose a new streaming approach which computes the LIS using a hierarchy structure of the stream. Our algorithm achieves a $(1 - \frac{T-b}{T+b})$-approximation to the length of the LIS in the worst case, where the parameter b controls both the perturbation noise to achieve the desired level of privacy and the accuracy. Finally, we discussion about possible extensions of our solutions to address time-series stream monitoring and string matching problems. To the best of our knowledge, we are the first to investigate the problem of privately computing the longest increasing subsequence.

1.2.4 Group Route Query (Chapter 6)

In this chapter, we propose a novel query, named *Optimal Group Route* (OGR) [11], which enables users to jointly plan an itinerary (i.e. a route) within their mobility constraints such that the overall preference on the locations visited by the group is maximized. The need for this type of queries naturally arise when a group of users wish to plan a group trip or a group outing together, each with certain mobility constraints and preferences on places to visit. Consider a group of users who would like to visit some categories of locations (POIs) together. The example categories are museums, coffee shops, libraries, etc. We assume that each user can specify his/her starting and ending points and a maximum distance he/she is willing to travel. Each user specifies a preference for each category of POI that she/he wishes to visit. Furthermore, a weight is assigned to each user to represent her/his influence in deciding the itinerary. Consider a representation of the spatial database as a location graph, where each node represents a location and an edge determines the connection between two locations. The goal of our optimal group route query is to find a route for each user such that: 1) the length of each route is under the individual distance threshold (mobility constraint), and 2) the overall weighted preference on the POIs visited together by the users is maximized. We first assume a scenario where users are interested only on the set of the specified categories regardless the order in which they are visited. Then, we extend our solutions to take into account user's specified order constraints on the categories (e.g. dinner followed by movie for a group outing).

We believe that this work advances research in traditional location-based services by introducing the social component in route recommendations and providing a balance between the users' social needs as well as their individual preferences. The ability of solving such group route query can greatly benefit many existing and emerging tools that allow users to share route information (e.g. Uber, Waze) and plan group outings or trips (e.g. QuickCliqs). For example, our approach enables users to construct a group itinerary which can be in turn used to

obtain driving directions. In this way, friends by sharing their preference and mobility constraints can easily obtain the directions to a place to meet. In a ride sharing setting, our solution not only enables users to find a common route but also could be used to match users in sharing the ride by considering user's preferences. Furthermore, by enabling users to share their routes, our group query could beneficially affect the traffic condition and environment by reducing the traffic and pollution in major cities.

In this chapter, we formally define the Optimal Group Route problem (OGR) and show the hardness of OGR by exploiting the connection with the Hamiltonian path problem. To solve this problem, we propose an exact dynamic programming algorithm which carefully constructs the itinerary by exploring a limited number of paths in the graph. Our algorithm runs exponentially with k, where k denotes the number of categories specified by the users. We observe that in practice k assumes small values with respect to the size of the graph, which allows our algorithm to compute the optimal solution efficiently. Furthermore, we propose two approximation algorithms with bounded approximation ratio. The first approach uses a scaled dynamic programming technique and achieves a $(1-\epsilon)$ approximation for the optimal itinerary. The second solution greedily constructs the itinerary achieving a $1/k$ approximation. We also show how to extend our solutions to consider POIs with different quality/popularity, order constraints on the required categories, and different problem relaxations (e.g. multi-objective utility).

1.3 Organization

The rest of the dissertation is organized as follows. In Chapter 2, we report the most related works to the problem discussed in this dissertation. In Chapter 3, we illustrate our contributions in privacy preserving pattern mining and in Chapter 4 we provide a concrete application setting where our solutions are applied for solving the problem of privacy preserving record linkage. We conclude our

contribution on the privacy aspects in Chapter 5 where we present our privacy preserving solution for statistics over data streams. In Chapter 6, we present the social component of our research by illustrating our work on group routing query. In Chapter 7, we finally summarize the contributions of this dissertation and state future research directions.

Chapter 2

Related Works

In this chapter, we summarize the related works most relevant to the contributions presented in this thesis.

2.1 Differential Privacy

Differential privacy [29] is a recent notion of privacy that aims to protect the disclosure of information when statistical data are released. The differential privacy mechanism guarantees that the computation returned by a randomized algorithm is insensitive to the change in any particular individual record in the input data.

Definition 2.1 (Differential Privacy [29])**.** *A privacy mechanism M satisfies ϵ-differential privacy if for any two input sets (databases) D_A and D_B with symmetric difference of one (neighboring databases), and for any set of outcomes $S \subseteq Range(M)$, the following inequality holds:*

$$Pr[M(D_A) \in S] \leq e^{\epsilon} \times Pr[M(D_B) \in S] \tag{2.1}$$

The parameter ϵ is the *privacy parameter* (also known as privacy budget) which defines the privacy level of the mechanism. Higher values of ϵ lead to lower

level of privacy, while smaller values pose a stronger privacy guarantee. From Definition 2.1, a mechanism M that satisfies differential privacy addresses the concerns that any user may have when the output of the mechanism is revealed. In fact, the output distribution of M would not significantly change even if the participant removes her/his data from the dataset in input. Definition 2.1 refers also to **user-level** differential privacy, since typically a user contribute to only one record in the database.

Such privacy notion is very strong, since it is a statistical property about the behavior of the mechanism M and therefore is independent of the computational power and auxiliary information available to the adversary/user. This is a crucial turning point with respect to previous privacy model (e.g. k-anonymity), where the adversary background knowledge may preclude the user's privacy.

2.1.1 Achieving Differential Privacy

To achieve differential privacy one well established technique is the *Laplace Mechanism* [32]. This strategy is based on the concept of *global sensitivity* [32] of the function to compute.

Definition 2.2 (Global Sensitivity [32])**.** *For any two neighboring databases D_A and D_B, the global sensitivity for any function $F : D \to \mathbb{R}^n$ is defined as:*

$$GS(F) := \max_{D_A, D_B} \|F(D_A) - F(D_B)\|_1 \tag{2.2}$$

For example, consider a function F that asks the following query: *"How many rows have property P? "*. For this function we have that $GS(F) = 1$, in fact by adding or removing one record, F can change at most by 1.

The Laplace mechanism is used in our work to construct differentially private algorithms, so we briefly discuss it below. Let F be a function, and ϵ be the privacy parameter, then by adding noise to the result $F(D)$ we obtain a differential privacy mechanism. The noise is generated from a Laplace distribution with probability

density function $pdf(x|\lambda) = \frac{1}{2\lambda}e^{-|x|/\lambda}$, where the parameter λ is determined by ϵ and $GS(F)$.

Theorem 2.3 (Laplace Mechanism [32]). *For any function $F : D \rightarrow \mathbb{R}^n$, the mechanism $M(D)$ that returns:*

$$M(D) = F(D) + Lap(GS(F)/\epsilon) \tag{2.3}$$

achieves ϵ-differential privacy.

Two composition properties are extensively used when multiple differential privacy computations are combined. These two properties are known as *sequential* and *parallel* compositions [57].

Theorem 2.4 (Sequential Composition [57]). *Let M_i be a non-interactive privacy mechanism which provides ϵ_i-differential privacy. Then, a sequence of $M_i(D)$ over the database D provides $(\sum_i \epsilon_i)$-differential privacy.*

Theorem 2.5 (Parallel Composition [57]). *Let M_i be a non-interactive privacy mechanism which provides ϵ_i-differential privacy. Then, a sequence of $M_i(D)$ over disjoint subsets of database D provides $(\max_i \epsilon_i)$-differential privacy.*

Additional statistical properties that we use in this dissertation are reported in Appendix 8.1

2.1.2 Event-level privacy

In the streaming setting, due to the dynamics of the data, the classical differential privacy notion has been redefined such that the privacy is guaranteed at *event-level* [31, 34, 33, 19]. In other words, the privacy goal is to protect the presence or absence of any single event in the stream. The formal definition of the differential privacy notion adopted in our streaming work is reported below.

Definition 2.6 (Differential Privacy [19, 33])**.** *Two streams σ and σ' of the same length are neighboring streams if they differ exactly in one element at time t. A privacy mechanism M gives ϵ-differential privacy if for any two neighboring streams σ, σ', and for any set of outcomes $S \subseteq Range(M)$, the following inequality holds:*

$$Pr[M(\sigma) \in S] \leq e^{\epsilon} \times Pr[M(\sigma') \in S] \tag{2.4}$$

Compared to Definition 2.1, the only differences here is the fact that the indistinguishability is guaranteed for any element in the stream in input. In fact, in this case the output distribution should be roughly the same whenever or not a single event is present in the input stream. In this way the adversary cannot distinguish if any single event is in the user's stream. Hence, the privacy is provided at event level.

2.2 Privacy Preserving Frequent Pattern Mining

In the data mining community, the pattern mining problem is often associated with patterns in the form of itemsets. Despite the wide range of algorithmic solutions to this problem, only a few approaches [6, 54, 79] study the mining of frequent patterns with differential privacy. However, due to the nature of the patterns, these approaches are not suitable for mining sequential patterns.

For sequential data, the problem of mining frequent patterns is commonly associated with mining trajectory data. Recently a variety of solutions [1, 5, 8, 68, 77] have been proposed for publishing privacy preserving trajectory data. Abul et al. [1] proposed the notion of (k, δ)-anonymity, which is based on the idea that given a location imprecision δ, the moving objects are indistinguishable if at least k of them are coexisting in the same cylinder of radius δ. In the work in [5], the authors proposed a technique to achieve anonymity in spatiotemporal datasets using spatial generalization and k-anonymity. Another extension of the k-anonymity for moving objects have been proposed by Yarovoy et al. [77]. Terrovitis and

Mamoulis [68] proposed a suppression technique whenever a privacy leak occurs, where the leakage is defined as the ability of an adversary of inferring the presence of a location using a set of spatial projections. Although all these techniques have been shown to be effective in several scenarios, their privacy models are not able to provide formal guarantees of privacy.

Under differential privacy, only two techniques have been proposed [22, 23] to tackle the problem of sequential pattern mining. The first approach proposed in [23] uses a differentially private prefix tree to partition the string data and to release a sanitized dataset from which frequent sequential patterns can be mined. Recently Chen et al. [22] proposed a technique based on variable length n-grams and a Markov model to publish a sanitized dataset. This method initially truncates the dataset by keeping only the first l_{max} symbols of the strings in input. Successively this truncated dataset is used to compute the frequency of the variable length patterns (n-grams) up to a fixed length n. Then the information about these patterns is used in a Markov model to release a sanitized dataset.

2.3 Privacy Preserving Record Linkage

There is a broad variety of strategies proposed by the scientific community to tackle the privacy preserving record linkage problem, which differ in privacy notions, protocol models, and the type of objects to be matched. In the following, we distinguish these techniques in three major categories: *secure transformations*, *Secure Multiparty Computation*, and *hybrid methods*.

Secure transformation techniques aim to perform the linkage after some transformations have been applied to the original data. The typical scenario involves three parties, where two parties have the data, and using secure transformation techniques they first transform the original data and the sent the transformed records to a third party whose task is to perform the matching. In this framework, to achieve privacy and security two major strategies have been proposed:

hashing and embedding. Approaches based on hashing functions try to match strings by hashing the original data and computing similarity measures after the hash functions are applied. Although these techniques are quite popular, they do not provide a formal bound on the distance in the hashed space so their usability is mostly restricted to exact matching. These techniques are characterized by the collision rate which defines the number of elements that will be map in the same value and poses a trade-off between utility and security. Furthermore, hash functions are deterministic which makes these techniques subject to dictionary attacks. In fact an adversary could mount a dictionary attack and by looking to the record that match up with the one hashed it could infer the original records. Examples of hashing techniques are Bloom filter [65], q-grams hashing [24], and TFIDF Hashing [3].

Embedding techniques instead map the original record in a vector space where the distance is generally preserved up to a distortion factor. A recent example of the embedding strategy for record linkage is the approach proposed by Scannepieco et. al. [64] that uses SparseMap [43] to embed strings into a vector space and perform matching in this new space. The core of this approach relies on the Lipschitz embedding [16, 46]. The mapping procedure projects each original data point s into a new space using a base $\mathcal{B} = \{A_1, A_2, \ldots, A_k\}$ which is a set of subsets A_i of the universe of elements. Each point s is mapped via the embedding function ρ into a vector $\bar{s} \in \mathbb{R}^k$, where each coordinate is computed as $\bar{s}_i = \min_{x \in A_i}\{d_{Edit}(x, s)\}$, for $i = 1, 2, \ldots, k$. The distance between vectors in the embedded space is measured by the Euclidean distance d', while the distance metric in the original space is the Edit distance d_{Edit}. The basic approach guarantees privacy since only a base of random strings is shared between the parties. However, to improve the performance of this approach, the authors in [64] proposed several heuristics that aim to carefully select the strings in the base to preserve the distance. Although these techniques improve the performance, they may disclose sensitive information about the single records. Indeed, as the proto-

col is designed, the shared base is optimized according to the data of one party. Therefore if the other party is malicious, it may break the privacy by inferring the original data from the structure of the base.

A key feature of embedding approaches is that the distance in the original space can be placed in relationship with the distance in the new space depending on the distortion induced by the embedding map. Unfortunately, bounding the distortion induced by some embedding may be technically challenging. Moreover, general embedding functions are computationally expensive to apply and some heuristic are needed. For these reasons, in this paper we present a complete study of the distance distortion both analytically and experimentally.

Secure Multiparty Computation (SMC) techniques cast the record linkage problem into a secure communication framework. In this setting, several parties are involved in the protocol where the communication is done using cryptography techniques. The key idea is that the computation itself should reveal no more than whatever may be revealed by examining the input and output of each party. An important theoretical result in the cryptographic area shows that any computational functions can be computed in this setting [76]. Motivate by this, several works have been proposed in the literature. For example, when the exact match is considered, the record linkage problem can be interpreted as a set intersection problem [**?**]. [**?**] gives a review of SMC approaches for privacy preserving data mining. While in principle the private record linkage problem can be solved using SMC and cryptography, the computational and communication cost is not practical in real applications.

Hybrid methods combine anonymization or secure transformation techniques with SMC techniques with the aim of reducing SMC cost. Inan et al. [44] proposed a strategy based on SMC and sanitization to achieve a trade-off between privacy and utility. This work has been further extended in [45, **?**] by differentially private blocking followed by SMC techniques for matching record pairs in corresponding blocks avoiding the comparison among all the record pairs. An-

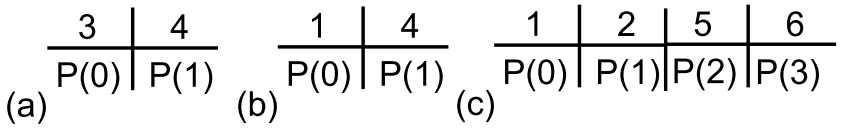


Figure 2.1: Running example of the Patience Sorting algorithm over the stream $\sigma = 3, 4, 1, 2, 5, 7, 6$.

other work which employs a blocking approach is [74]. The solution consists in two steps, first the records are transformed into real vectors and second the transformed records are matched using a secure record linkage technique. While hybrid techniques provide a good trade-off between privacy and accuracy, the SMC step still involves high computational cost and the impact of the blocking on the linkage accuracy is not clearly understood.

2.4 Stream

2.4.1 Longest Increasing Subsequence

The problem of computing the LIS has received much attention in the streaming setting (see [4] for a survey of results), where the sequence σ is given an element at a time. In such model, data arrive continuously and at every time i algorithmic solutions are required to report $LIS(\sigma[0, i])$ by using a small amount of memory and performing only few passes over the stream. In the rest of the section, we briefly summarize the non-private techniques present in literature by categorizing them as *exact* and *approximate* solutions.

Exact Solution. The study of the longest increasing subsequence (LIS) in the streaming setting was initiated by Liben-Nowell et al. in [55], where the authors developed an exact one pass algorithm that requires $O(k)$ space for deciding if the length of longest increasing subsequence is at least k. In addition to this technique, the classical algorithm for computing the LIS is based on the Patience Sorting pro-

Algorithm 1 Patience Sorting

1: **procedure** PATIENCE SORTING(σ)
Input: event stream σ
Output: $LIS(\sigma)$ length of the longest increasing subsequence

2: $P(j) \leftarrow \emptyset$ for $j = 0, 1, \ldots, m-1$
3: **for** (any new element $\sigma(i)$) **do**
4: Find the largest $P(j)$ such that $P(j) \leq \sigma(i)$
5: $P(j+1) = \sigma(i)$
6: **Output** the largest j such that $P(j) \neq \emptyset$
7: **end for**
8: **end procedure**

cedure [41]. This approach can be interpreted as a one pass streaming algorithm for computing the exact LIS in $O(T)$ space and it requires $O(\log LIS(\sigma))$ update time. Since we use this approach to build our solutions, we briefly describe this algorithm here.

In the Patience Sorting procedure, the length of the longest increasing subsequence is computed using a set of sorted *piles* $P(0) < P(1) < \cdots < P(m)$ each storing an element of the stream σ. For any new element $\sigma(i)$ that appears in the stream, the algorithm places $\sigma(i)$ in the leftmost pile $P(j)$ such that $P(j) > \sigma(i)$. The number of non empty piles represents the length of the LIS at any time point. An overview of the Patience Sorting algorithm is illustrated in Algorithm 1. Below, we describe a running example of this algorithm.

Example 2.7. *Let $\sigma = 3, 4, 1, 2, 5, 7, 6$ be a stream in input. The algorithm starts with a set of empty piles $P(j)$ for $j = 0, \ldots, m-1$. When the first element arrives in the stream it is placed in the first pile $P(0)$. After the arrival of the second element, the situation in the piles is illustrated in Figure 2.1 (a). The number of piles denotes the length of the longest increasing subsequence at each time. Therefore, in this case the length of the LIS is two. When the third element*

$\sigma(2) = 1$ arrives in the stream, the algorithm places this element in $P(0)$, as shown in Figure 2.1 (b). Following the steps of the algorithm, the final set of piles is reported in Figure 2.1 (c). At the end of the stream the length of the longest increasing subsequence is four.

Despite the simplicity of this procedure, the Patience Sorting algorithm is optimal from the space complexity perspective. In fact, Gopalan et al. [38] showed a space lower bound of $\Omega(n)$ for any randomized algorithm that computes the LIS exactly.

Approximate Solution. In [38] the authors proposed a $(1+\epsilon)$-approximation for the LIS computation using $O(\sqrt{T/\epsilon})$ space. A series of works have been developed to estimate the length of the LIS using the number of inverted elements in the stream. In this direction, Ajtai et al. [2] proposed a $(1+\epsilon)$-approximation which requires $O(\frac{1}{\epsilon}\log\log T)$ space to estimate the number of inverted pairs. Later this result has been improved by Gupta and Zane [39]. Cormode et al. [26] proposed a series of algorithmic solutions based on distance preserving embedding. Recently in [63], the authors investigated the problem of computing the LIS in asymmetric edit distance setting.

2.5 Spatial Queries in Location-based Services

The search of optimal routes on graphs has its roots in the "Orienteering Problem" (OP). As in the sport game, a user starting from a point tries to visit as many nodes as possible within a given time constraint. Our proposed problem differs from the OP in many aspects. While OP is limited to a single user, in our formulation we are interested in computing a shared route for multiple users. Furthermore, in OP the profit in visiting a node is fixed, while in our problem such value depends on the users' preferences and on the sequence of nodes already covered, since the overall score of the route is computed on a set of unique categories. We refer the

interested readers to [71] for a survey on the OP problem.

Li et al. [52] are the first to introduce the problem of finding route on spatial databases. In that work, the authors proposed a new problem called Trip Planning Query (TPQ), where each object in the database is associated with a location and a category label. Given a set of categories in input, a starting node and an ending node, the TPQ problem aims to find the shortest route from the starting to the ending node that passes through at least one point for each category. The authors in [52] showed that the TPQ problem is NP-hard, and they proposed a series of approximation solutions to tackle the problem. Compared to TPQ, our problem considers distance constraints and our goal is to maximize the profit for a group of users on the locations that are jointly visited.

Kanza et al. [49] proposed a new query problem, where the length of the route is bounded by a constraint and the goal is to maximize the profit in covering the categories of the nodes in the route between a given start and end point. The major differences between this approach and our work can be outlined as follows. First, in [49] an exact number of categories must be visited while in our case any subsets could represent a possible candidate. Second, in our case the score associated to each category depends on the user preferences while in Kanza's work the categories have the same score. Furthermore, we consider multiple users and therefore the starting and ending points of the optimal shared route are not fixed as in [49] but depend on the locations where the users could meet. In successive works, Kanza et al. [47, 48] extended their approach by considering an interactive setting and introducing order constraints.

Sharifzadeh et al. [66] introduced a new extension of the TPQ problem, named Optimal Sequenced Route (OSR), where the goal is to find the shortest route from a given node that passes through an ordered sequence of locations. The authors proposed a series of pruning techniques to discard those locations that cannot be part of the optimal route. Chen et al. [20] proposed an extension of TPQ and OSR which considers the search of multi-rule partial sequenced routes (MRPSR).

The authors showed that the MRPSR provides a unified framework that subsumes TPQ and OSR. In the original paper a series of heuristic algorithms have been developed to solve MRPSR queries. Recently, another extension of TPQ has been proposed by Li et al. [53]. In this formulation, the goal consists in computing the shortest route that covers a user defined set of categories with partial order constraints. The authors proposed two approaches namely backward and forward search to efficiently compute the optimal route.

Recently, Cao et al. [18] proposed a new route problem called Keyword-aware Optimal Route Search (KOR), which given a pair of nodes representing a start and end location in a graph, consists in finding the route that connects those two points such that a set of user-specified keywords is covered, a specified budget constraint is satisfied, and an objective score of the route is optimized. The authors in [18] proved the hardness of such formulation, and developed a series of approximation algorithms to solve the KOR problem. Our group trip query differs from both OSR and KOR in several aspects. We consider multiple users in the query, and we find the route of bounded length that maximizes the profit for the categories covered. Therefore, the solutions for OSR and KOR are not suitable in our setting.

B. Roy et al. [62] investigated the problem of computing an itinerary in an interactive way, where the user can provide feedback on the selected POIs to improve the recommended itinerary.

When multiple users are involved in the route query, Hashem et al. [42] proposed a new problem called Group Trip Planning Query (GTP). Given a set of users with their start and end point in input, and a set of location categories, the GTP problem consists in finding a set of nodes belonging to the specified categories that minimizes the total traveled distance by the group. It has been shown by the authors in [42] that the GTP problem is related to the Group Nearest Neighbor query (GNN). Based on this observation, the authors proposed a series of heuristic approach to solve the GTP problem. Our approach is different from GTP since we consider a preference score associated to the categories rather than considering all

the categories in the same way. Furthermore, we are interested in finding the best route in terms of preference score rather than distance.

Chapter 3

Frequent Pattern Mining of Sequential Data

3.1 Problem Definition

In this work, we are interested in mining *sequential patterns* that are in the form of a sequence. A pattern p of length n is represented as a sequence of symbols $p = a_0 a_1 \cdots a_{n-1}$ where each symbol a_i belongs to a finite alphabet Σ. We denote the length of p with $|p|$. In the sequel, we also refer to these sequential patterns as *strings*. Furthermore, we say that a pattern p *occurs* at position i in a string x if there exists $j \in [0, |x| - |p|]$ such that $x_{j+i} = a_i$ for $i = 0, \ldots, |p| - 1$. In other words, the substring $x[j, j + |p| - 1]$ matches the pattern p. We introduce the concept of frequency of a pattern as follows.

Definition 3.1 (Frequency)**.** *For any pattern p we denote by $f_x(p)$ the number of occurrences of p in x. When a set of N strings $D = \{x^0, x^1, \ldots, x^{N-1}\}$ is given as input, we define by $f_p := \sum_{i=0}^{N-1} f_{x^i}(p)$ the frequency of the pattern p in D.*

The definition of frequency based on the number of occurrences of a pattern allows us to capture those repetitive patterns that appear multiple times within

the same string. This is crucial for a variety of domains such as time-series data, where the use of the frequency of the patterns within the same series is used to predict the near future values. Our mining problem is formalized below.

Problem 3.2 (Top-k mining problem)**.** *Given a positive integer k, and a range of pattern lengths I, report the list $\mathcal{F}_I$ of the top-k most frequent patterns of variable length on the input set D.*

In this paper, we consider both *substring* and *prefix* patterns. A substring pattern for a dataset D is represented as any sequence of symbols that are occurring in D. On the other hand, a prefix pattern p is a substring pattern such that there exist some strings in D starting with p. Furthermore, the cardinality of this set represents the occurrence of p in D.

Example 3.3. *Let $D = \{ababbaa, abab, babba\}$ be an input datset. The pattern $p_1 = aba$ is a prefix pattern for D since both the first and the second string start with p_1. The pattern $p_2 = bba$ is only a substring pattern for D, since no input records start with it. Both these patterns occur twice in D.*

In the rest of the paper, the term pattern will refer to substring pattern if not specified otherwise.

3.1.1 Problem Challenges

In the differentially private mining of frequent sequential patterns there are two major challenges. First, the mining of these patterns is computationally intense. Indeed, the possible number of patterns grows exponentially with the length of the patterns, which makes the mining process inefficient if done naively. Second, the count of occurrences of patterns has high sensitivity, which means that a large amount of perturbation noise is needed to guarantee differential privacy. A formal analysis for the sensitivity of counting occurrences is reported below.

Lemma 3.4 (Sensitivity for Counting Occurrences)**.** *Let D be an input dataset with maximum string length l_{max}, then for a query $q = [p_1, p_2, \ldots, p_n]$ which for each pattern p_i computes the number of occurrences in D, the sensitivity $GS(q)$ of q is at most l_{max}.*

Proof. For any string x, the maximum number of occurrences that it can contribute in the query q is $|x|$. In fact, there are at most $|x|$ patterns (either the same repeated or different patterns) that can occur in x. Therefore, it follows that for any neighbour dataset D' obtained by removing or adding a single string from D the answer for q can change at most by l_{max}. □

Lemma 3.4 states that the sensitivity of counting occurrences is as high as the length of the longest string in the dataset. Therefore, even if only one very long string is present in the input dataset, all the frequency of the patterns have to be perturbed by a large amount of noise which can drastically reduce the utility.

3.2 A Baseline Solution

Lemma 3.4 points out that the sensitivity for counting the occurrences of patterns is bounded by the maximum length of the strings in the input dataset. This fact can result in loss of accuracy for the mined patterns since even the presence of one very long string in the input forces a large amount of noise. However, in real world applications we can observe that the majority of the strings in the data are short and only few of them are very long. Consider for example the Anonymous Web Data `MSNBC`, where each sequence in the dataset corresponds to page views of a user during that twenty-four hour period, the average length for the strings is only 4.7 symbols while the maximum length is 14975. Intuitively, since most of the strings are short it is reasonable to assume that the occurrences for the frequent patterns are probably mostly captured by the short strings rather than the long ones. This observation motivates us to consider an alternative way to

mine the patterns. In particular, we want to privilege the short strings over the long ones by processing the strings in the datasets starting from their prefix. A baseline approach has been firstly proposed for mining trajectory data based on prefix tree is reported in [21, 23]. We further extended such approach and applied in a different setting in [14]. We briefly describe this solution in the section below.

3.2.1 Prefix Tree Algorithm

Algorithm Description. The construction of the prefix tree can be summarized as follows. Starting from the root node, the database is partitioned by extending the prefix of the current node using the procedure in Algorithm 2. Given a current node representing the prefix ω, for every symbol a in the alphabet Σ, a new node associated with the prefix ωa is attached to the tree only if the string ωa is a frequent prefix. To determine if a prefix is frequent, a counting query is issued on the partition of the dataset represented by the current node and the real count is perturbed by Laplace noise to guarantee differential privacy. In this process, only partitions with frequent prefixes ($count > \theta$) are further refined. The allocation of the budget at each level in the tree is performed at line 11 in Algorithm 2. In our approach, we propose several strategies to allocate the privacy budget: *linear allocation*, *exponential allocation*, *adaptive*, and *hybrid*. Details about these strategies are presented later in this section. After we partition the data, we traverse the prefix tree and apply the consistency constraints for each root-to-leaf path as in [21]. Once the consistency constraints are enforced, we identify a list of frequent patterns by traversing the tree. As a final result, we return the top-k patterns sorted by their noisy frequencies.

Example 3.5. *Consider the prefix tree illustrated in Figure 5.2. Each node in $\mathcal{T}$ identifies a partition. For example, the first internal node labeled with "*`a`*" is associated with a partition in dataset containing the records "*`aaba`*" and "*`abca`*" of index 1 and 5 respectively. Furthermore, the frequency for the pattern aa is*

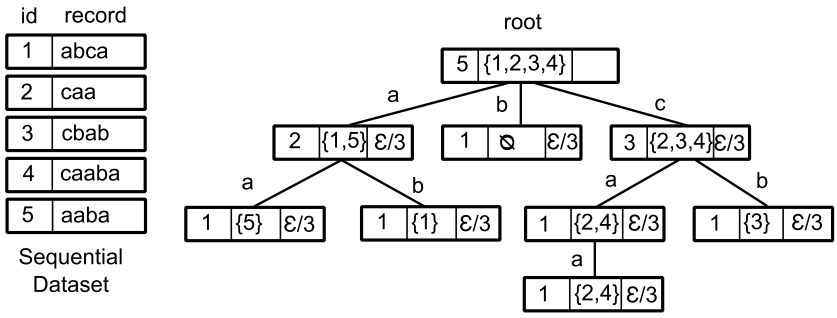


Figure 3.1: Example of prefix tree. Each node has a noisy count, a partition of strings sharing the same prefix, and a privacy budget.

computed by summing the counts of the two prefixes aa *and* caa *associated to the partitions* $\{5\}$ *and* $\{2, 4\}$ *respectively.. Therefore, the final noisy count for* aa *is* $2 = 1 + 1$.

Budget Allocation Strategies.. To allocate the privacy budget during the partitioning process, we propose several allocation strategies.

1. **Linear:** Each node in the tree is assigned with the same amount of budget. This strategy treats all the nodes in the same way which could result to be ineffective for capturing long prefixes.

2. **Exponential:** At level i in the tree, each node is assigned with a privacy budget which is double the amount of its parent. This strategy is motivated by the fact that prefixes in the top levels of the tree have a considerable higher counts of those down in the tree. Hence, we can afford to spend less privacy budget in the first part of the root-to-leaf path and use more budget later in the exploration.

3. **Adaptive:** This strategy is an adaptation of the exponential strategy, where the entire remaining budget on the path is spent on the next counting query if the current node represents a non frequent prefix. In this way, we minimize the amount of privacy budget that is not used in the construction process.

Algorithm 2 Private Prefix-Tree Partitioner

procedure PPT PART(D, ϵ)
Input: dataset D; privacy parameter ϵ
Output: $\mathcal{T}$ private Prefix-Tree

$\mathcal{T}$ formed by the $root$
Use a queue Q
$Q \leftarrow root$
while (Q is not empty) **do**
  $node \leftarrow Q.remove$
  if ($node.h < h_{MAX}$) **then**
    for (every symbol a in the alphabet Σ) **do**
      $\omega \leftarrow$ (path from r to node) + a
      $P \leftarrow \{x \in node.set$ s. t. ω is a prefix of $x\}$
      $\tilde{\epsilon} \leftarrow$ ALLOCATE BUDGET($node$)
      $\tilde{c}(\omega a) \leftarrow |P| + Lap(1/\tilde{\epsilon})$
      if ($\tilde{c}(\omega a) > \theta$) **then** ▷ Non empty node
        $cur.set \leftarrow P$, $cur.epsilon \leftarrow \tilde{\epsilon}$
        $cur.label \leftarrow \omega$, $cur.h \leftarrow node.h + 1$
        $cur.budget \leftarrow cur.budget + cur.epsilon$
        Attach cur as a child of $node$ in $\mathcal{T}$
        $Q \leftarrow cur$
      end if
    end for
  end if
end while
return $\mathcal{T}$
end procedure

4. **Hybrid:** This strategy is a combination of the previous strategies, where the total budget is distributed in the tree according to q_{max}. In order to increases the chances of extending the tree and capture longer prefixes, we split the total privacy budget in two parts among the levels in the tree. The first half is reserved for the nodes on the first q_{max} levels of the tree, where

for each node the budget is allocated a linear fashion. The remaining part of the budget is allocated for the lower levels of the tree using the adaptive strategy.

Privacy Analysis.. In the following, we show that our prefix-tree based approach achieves ϵ-differential privacy, where ϵ denotes the level of privacy required by the users.

Lemma 3.6. *For every root-to-leaf path in the prefix tree, the total privacy budget used is at most* ϵ.

Proof. Let $\pi = \nu_0\nu_1 \dots \nu_h$ be a root-to-leaf path in the prefix tree where $h \leq h_{MAX}$. We want to show that for every allocation strategy proposed, the total privacy budget does not exceed ϵ. Therefore, denoting ϵ_i the budget used for node ν_i, we want to prove that $\sum_{i=0}^{h} \epsilon_i \leq \epsilon$ for every budget allocation strategy.

1. **Linear:** for a node ν_i in the path we have $\epsilon_i = \epsilon/h_{MAX}$. Since $h \leq h_{MAX}$ it follows that $\sum_{i=0}^{h} \epsilon_i \leq \epsilon$.

2. **Exponential:** for a node ν_i in the path we have $\epsilon_i = 2\epsilon_{i-1}$, $i = 1, \dots, h$ where $\epsilon_0 = \frac{\epsilon}{2^{h_{MAX}+1}-1}$. Therefore,

$$\sum_{i=0}^{h} \epsilon_i = \frac{\epsilon \sum_{i=0}^{h} 2^i}{2^{h_{MAX}+1} - 1} = \frac{(2^{h+1} - 1)\epsilon}{2^{h_{MAX}+1} - 1} \leq \epsilon \tag{3.1}$$

3. **Adaptive:** this strategy is similar to the previous one. The only difference is that in this case we spend the remaining privacy budget on the next counting query if the current node is not frequent. This test preserves the privacy since it is done on noisy counts.

4. **Hybrid:** for a node ν_i in the path, we have ϵ_i as follows:

$$\epsilon_i := \begin{cases} \frac{\epsilon(i+1)}{q_{max}(q_{max}+1)} & \text{if } 0 < i < q_{max} \\ \frac{\epsilon 2^{i-q_{max}-1}}{2(2^{h_{MAX}-q_{max}}-1)} & \text{otherwise} \end{cases}$$

For the first q_{max} levels only half of the total privacy budget is used:

$$\sum_{i=0}^{q_{max}-1} \epsilon_i = \frac{\epsilon \sum_{i=0}^{q_{max}-1}(i+1)}{q_{max}(q_{max}+1)} = \epsilon/2 \tag{3.2}$$

For the remaining nodes in the path, the allocation of the budget follows the exponential strategy, therefore:

$$\begin{aligned}\sum_{i=q_{max}}^{h} \epsilon_i =& \frac{\epsilon \sum_{i=q_{max}}^{h} 2^{i-q_{max}-1}}{2(2^{h_{MAX}-q_{max}}-1)} = \\ =& \frac{\epsilon}{2} \cdot \frac{2^{h-q_{max}}-1}{2^{h_{MAX}-q_{max}}-1} \leq \epsilon/2\end{aligned} \tag{3.3}$$

Summing up the results from Equation (3.2) and Equation (3.3), the hybrid strategy on any root-to-leaf path uses at most ϵ privacy budget.

This concludes the proof of the Lemma. □

Using the parallel composition property in [57] we have the following result.

Theorem 3.7 (Prefix-Tree ϵ-privacy)**.** *The Prefix-Tree Miner guarantees ϵ-differential privacy.*

Proof. All the partitions produced by Algorithm 2 on the same level of the tree are disjoint since they correspond to strings with different prefixes. Therefore by the *parallel composition* property in Theorem 2.5, the overall privacy is bounded by the max of the total privacy budget used on any root-to-leaf path in the Prefix Tree. Lemma 3.6 shows that all the allocation strategies use at most ϵ privacy budget on any root-to-leaf path in the tree. Thus the Prefix Tree approach satisfies ϵ-differential privacy. □

Complexity Analysis.. Algorithm 2 has running time proportional to the number of nodes in the prefix tree $\mathcal{T}$. By using a similar analysis as in [21], it can be shown that our mining approach requires $O(N|\Sigma|^{h_{MAX}+1})$ operations, where N is the size of the dataset, Σ is the alphabet, and h_{MAX} is the maximum depth in the tree.

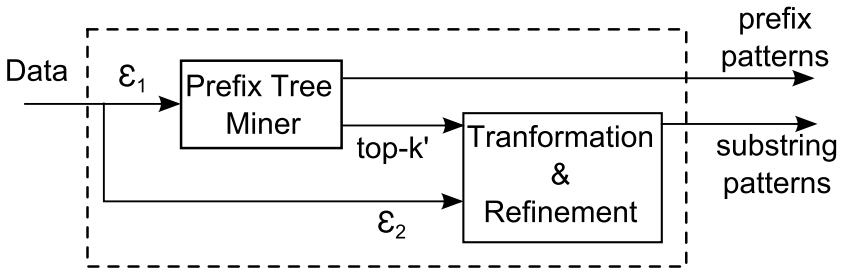


Figure 3.2: Overview of two-phase algorithm.

3.3 Two-Phase Algorithm

In this section, we present our two-phase algorithm to address the afore-mentioned challenges. An overview of our approach is illustrated in Figure 3.2. In the first phase, we use an enhanced **prefix tree miner** to release an ϵ_1-differentially private prefix tree. We will use this tree to first directly mine prefix patterns, and to second retrieve a set of top-k' patterns to use in the second phase. In the second phase, we use a **transformation and refinement** step to first construct a sketch of the dataset and second refine the count of the candidate patterns. We use the remaining ϵ_2 privacy budget to query the transformed dataset and retrieve the final counts.

Our strategy presents several advantages. First, the use of a prefix tree allows us to achieve high utility for mining frequent prefixes. Second, the use of a candidate set of patterns help us to limit the number of possible candidate frequent patterns to mine in the second phase. In fact, rather than considering the entire universe of patterns, we focus only on k' candidates. Third, we refine the count of the mined patterns in the top-k' by issuing counting queries on a transformed version of the dataset. In this way, we show that we can control the sensitivity of the counting query leading to a smaller amount of perturbation noise in reporting the final top-k patterns. In the rest of the section, we describe these two phases in details.

3.3.1 Model-based Prefix Tree Miner

Inspired by this technique, we introduce an enhanced prefix tree mining algorithm which uses statistical properties of the data to maximize the quality of the counts in the nodes of the tree. Our goal consists in using the information from a statistical model to calibrate the noise injected with the Laplace Mechanism so that we minimize the effect of the perturbation noise on the overall utility. First we develop a solution based on a sample of the real data, denoted as PT-Sample. Second, we propose an alternative solution to handle the case where this a priori information is not available, we call this algorithm PT-Dynamic.

Markov Model

An m-order Markov model is a statistical model that can be used to predict the frequency of strings. In particular, an m-order Markov model is based on the *Markov independence assumption* which states that the presence of a particular symbol in a sequence depends only on the previous m symbols. Formally, given a sequence of n observed symbols $X = X_1 X_2 \dots X_n$, and let $m < n$, using the Markov assumption we have that:

$$Pr[X_i = a|X] \approx Pr[X_i = a|X[i-m, i] = y[1, m]] \tag{3.4}$$

A stationary Markov model of order m is completely defined by its transition matrix $\Pi = (\pi(y[1, m], a))$ for $y[1, m] \in \Sigma^m$, where each π is transition probability defined as follows:

$$\pi(y[1, m], a) = Pr[X_i = a|X[i-m, i-1] = y[1, m]], m < i < n \tag{3.5}$$

Since the true model is generally unknown the transition probability can be estimated by using the maximum likelihood estimator as in [61].

$$\hat{\pi}(y[1, m], a) \approx \frac{f_D(y[1, m]a)}{f_D(y[1, m])} = \frac{f_{y[1,m]a}}{f_{y[1,m]}} \tag{3.6}$$

Algorithm 3 Private Prefix-Tree Sample Miner

procedure PT-SAMPLE(D, ϵ_1, S)
Input: dataset D; privacy parameter ϵ_1; Sample S
Output: $\mathcal{T}$ private Prefix-Tree

Construct a Markov Model of order m from the sample S
$\mathcal{T}$ formed by the $root$
Create a queue Q
$Q \leftarrow root$
while (Q is not empty) **do**
  $v \leftarrow Q.remove$
  for (every symbol a in the alphabet Σ) **do**
    Extend the prefix ω of the node v with the symbol a
    Create a possible child ν of v
    $\tilde{c}(\omega a) \approx \tilde{c}(\omega) \frac{f_S(\omega[1,m]a)}{f_S(\omega[1,m])}$ ▷ Model Estimate
    $\epsilon_\nu \leftarrow$ BUDGET ALLOCATION$(\nu, \lambda, \epsilon_{acc})$
    Calculate the noisy count for ν
    Decide to extend ν or mark it as a leaf node
    Add ν to Q
  end for
end while
Enforce consistency constraint on $\mathcal{T}$
end procedure

A Sample Based Model

In the construction of the prefix tree we employ the Laplace Mechanism combined with the estimated count from a Markov model constructed on a sample in input. In many scenarios, we can assume that together with the sensitive data we have some publicly available information that we can use in our mining process. Consider for example the case of medical data, many patients prefer to keep their privacy right on their data but it often occurs that a part of them opt to share or disclose their data with consent. Such information can be captured by a statistical

model and used in the construction process of the prefix tree.

One of the challenges in using the Laplace Mechanism consists on the fact that the noise injected to achieve ϵ_1-differential privacy depends only on the value of ϵ_1 and sensitivity of the query. Since the magnitude of the noise does not depend on the real answer, the mechanism turns out to be in favour of queries with a large count. Indeed, given a fixed amount of privacy budget ϵ_1 we will incur a larger error for those queries with smaller values of count. Therefore, if in our tree we want to guarantee overall ϵ_1-differential privacy, the use of an equal amount of privacy budget among the nodes will result in penalizing those with smaller counts. To mitigate this phenomena, we first construct a Markov model from a sample, and we use the model to obtain an estimated count for each node. Second, we use this information to properly decide the amount of noise to inject so that the overall prefix tree satisfies ϵ_1-differential privacy and the error introduced by the noise is reduced. For example, in any root-to-leaf path instead of using a fixed amount of privacy budget among the path, we can spend a smaller budget for nodes with large count and use this saving later in the path for nodes with smaller counts. Our procedure is illustrated in Algorithm 3.

Budget Allocation. We start assigning a privacy budget $\bar{\epsilon}$ for nodes associated to prefixes of length less than m (the length of the model) which assumes the default value of ϵ_1/h_{max}. This initialization step is motivated by the fact that for these nodes the model cannot provide a good estimated count, so it is better to use the Laplace Mechanism directly with parameter $\bar{\epsilon}$. Then in our top-down partitioning process, for a generic node ν with prefix ωa we first compute an estimated count using the model as in line 11 of Algorithm 3. Then when the estimated count $\tilde{c}(\nu) = \tilde{c}(\omega a)$ is returned, we compare it against a threshold value $\lambda = \alpha(|\tilde{D}|/|P_l|)$, where α is a constant, $|\tilde{D}|$ is the noisy version of the size of the dataset, and P_l is the set of prefixes in the tree of length l. The value of λ gives an indication about the average counts of the prefixes in the previous level in the tree, and it is used to decide the amount of privacy budget for the node ν.

Algorithm 4 Budget Allocation

1: **procedure** BUDGET ALLOCATION($\nu, \lambda, \epsilon_{acc}$)
Input: current node ν; threshold λ; privacy budget used till the parent of ν ϵ_{acc}
Output: ϵ_ν privacy for the node ν

2: $\tilde{c}(\nu) \leftarrow \nu.model\_count$
3: $\bar{\epsilon} \leftarrow (\epsilon_1 - \epsilon_{acc})/(h_{max} - \nu.h)$
4: **if** $(\tilde{c}(\nu) < \lambda)$ **then**
5: **return** $\bar{\epsilon}$
6: **else**
7: **return** $\bar{\epsilon}\frac{\lambda}{\tilde{c}(\nu)}$
8: **end if**
9: **end procedure**

The privacy budget allocation strategy is illustrated in Algorithm 4. The procedure computes for a node ν the amount of privacy budget to allocate. Given the amount of privacy budget allocated so far on the path from the root to the parent node of ν, the algorithm starts with a privacy budget $\bar{\epsilon}$ as in line 3. This value is defined as the privacy budget left on the path divided by the remaining level in the tree. Then if the estimated count $\tilde{c}(\nu)$ is smaller than the threshold λ this privacy parameter is directly returned, otherwise it will be scaled by a factor $\lambda/\tilde{c}(\nu)$, which is in the range $(0, 1)$. Note that if already the overall privacy budget has been consumed (i.e. $\epsilon_{acc} = \epsilon_1$), then no further privacy parameter ϵ_ν will be allocated.

Noisy Count Computation. After the privacy parameter ϵ_ν is determinate, the noisy count for the node ν is computed by perturbing the real count of ν with Laplace noise with parameter ϵ_ν. This perturbed count will be assigned as the final count for the prefix represented by the node ν if the count from the model is smaller than λ. Otherwise, we use as a final count the average between the noisy count and the count from the model. Notice that, if we could have an estimate of the error in our Markov model at this step, we could employ a weighted average

to determine the count. However, without this information we decide to use the simple average which from our experiments provides fair results.

Finally to decide if a node is a leaf or an internal node in the tree we compare its noisy count against the threshold value θ. If the current node is a leaf, we use up all remaining privacy budget to refine its count.

A Dynamic Model

In the previous section, we presented a construction process which uses the statistical properties from a sample of the data to calibrate the noise in the tree. However, in certain settings a sample of the data or background knowledge are not always available. In such cases we cannot rely on a priori information. In this section we show how to modify our previous technique so that during the tree construction process we can dynamically use the partial tree to construct a statical model. We call this algorithm PT-Dynamic. The idea of using some information about noisy data to improve the utility has been shown to be effective in reducing the relative error for count queries in the recent work of Xiao et al. [73].

Our strategy follows similar steps as in Algorithm 3, where now every time a new node is attached to the prefix tree the statistical information are updated. Therefore, now the Markov model is defined on the noisy frequencies generated during the construction of the prefix tree. In this case the estimated frequency of a prefix ωa is computed using the transition probability obtained from the noisy frequencies released in the previous levels of the tree as follows:

$$\tilde{c}(\omega a) \approx \tilde{c}(\omega)\frac{\hat{f}_D(y[1,m]a)}{\hat{f}_D(y[1,m])} \tag{3.7}$$

We can observe that for each new node added to the prefix tree, the only information that has to be updated are the frequencies $\hat{f}_D(y[1,m]a)$ and $\hat{f}_D(y[1,m])$ which can be efficiently computed.

3.3.2 Error Analysis for Prefix Tree

In this section, we investigate the error in the counts reported by the prefix tree for two types of patterns: *prefixes* and *substrings*. In particular, we would like our mining algorithms to be useful, that is, their output should well approximate the real count of the pattern (prefix and substring) on the input data. Below, we formally define the notion of utility for counting query.

Definition 3.8 ((ξ, δ)-Useful)**.** *A mining/counting algorithm $\mathcal{A}$ is (ξ, δ)-useful, if for any data input D and pattern p, with probability at least $1-\delta$, the approximate answer from $\mathcal{A}$ is within ξ from the real count of p.*

Prefix Patterns. The counts for the prefixes can be directly retrieved from the node in the prefix tree. Any prefix count query for a pattern p, can be answered using the prefix tree by returning the noisy count of the node with prefix label p. We quantify the error bound of the noisy frequency in the following Corollary.

Corollary 3.9 (Error Bound for prefix query)**.** *For any prefix count query for a pattern p, the noisy count $\tilde{c}(p)$ associated to the node in the tree having label p and privacy parameter ϵ_i, with probability at least $1-\delta$, the quantity $\xi = \|c(p) - \tilde{c}(p)\|_1$ is at most $O(\frac{1}{\epsilon_i} log \frac{1}{\delta})$.*

Proof. It follows from the tree construction, and the pdf of Laplace distribution. □

Substring Patterns. In our tree representation, the mining of substring patterns is a more challenging problem than mining prefixes. While there is a one to one correspondence between the frequent prefix pattern and the node in the tree, for frequent substring patterns this relationship is more complex. Indeed, the frequency of a pattern p is computed as the sum of the noisy count of the prefixes where p occurs as a suffix. Below we quantify the noise accumulated in this process, which will help us to understand the theoretical limitations of this approach.

Formally, let $\tilde{f}_p$ be the frequency released by the prefix tree for the pattern p, and denote by f_p its real frequency. Let $\mathcal{P}(p)$ denote the set of nodes in the tree having p as a suffix, then we can write $\tilde{f}_p$ as follows.

$$\tilde{f}_p = \sum_{\nu \in \mathcal{P}(p)} \tilde{c}(x) = \sum_{\nu \in \mathcal{P}(p)} c(\nu) + \sum_{\nu \in \mathcal{P}(p)} Lap(1/\epsilon_\nu) \tag{3.8}$$

In the construction of the tree, some prefixes are not extended due to the fact that their noisy counts do not pass the threshold value θ. Therefore, the sum of the counts for the prefixes in the tree containing p as a suffix can be upper bounded by the frequency of p in the real data f_p as follows.

$$\tilde{f}_p \leq f_p + \sum_{\nu \in \mathcal{P}(p)} Lap(1/\epsilon_\nu) \tag{3.9}$$

We denote the sum of these Laplace noises with the random variable $Y = \sum_{\nu \in \mathcal{P}(g)} Lap(1/\epsilon_\nu) = \sum_{i=0}^{n} Lap(1/\epsilon_i)$. Therefore, the variable Y determines the error in estimating the absolute value of the frequency $\xi = \|\tilde{f}_p - f_p\|_1$. Below, we characterized this quantity.

Corollary 3.10 (Error Bound for substring query)**.** *For any substring count query for the pattern p, the noisy count $\tilde{f}_p$ obtained by summing the noisy count of the nodes in the set $\mathcal{P}(p)$ of size n, with probability at least $1 - \delta$, the quantity $\xi = \|f_p - \tilde{f}_p\|_1$ is at most $O(\sqrt{\sum_{i=0}^{n-1} \frac{1}{\epsilon_i^2}} log\frac{1}{\delta})$.*

Proof. The proof follows from Corollary 8.2, where we choose $\nu = \sqrt{\sum_{i=0}^{n-1} b_i^2}\sqrt{2 \ln \frac{2}{\delta}}$. □

This result shows a distinction between the utility for prefix and substring patterns. As we expected the error for mining prefixes is small, while for substring we accumulate noises from multiple nodes. This motivate us to use the prefix tree to mine the prefixes directly and use a second phase to improve the final results for the substring patterns.

3.3.3 Transformation & Refinement

In the previous analysis we pointed out that counting the occurrences of substring patterns from the tree could incur a large error due to the sum of multiple perturbation noises. This could results in poor performance if we just mine the top-k in this phase. However, due to the nature of the distribution of the frequency for the frequent patterns, we can use the prefix tree to generate a candidate set of k' patterns with $k' > k$, which likely contains the real top-k patterns. Therefore, the goal of our second phase consists in finding the top-k patterns from the set of candidates.

In this phase, we refine the count of the patterns issuing a counting query, however due to its high sensitivity we could incur a large perturbation noise if it is applied on the original dataset directly, as shown in Lemma 3.4. Therefore, our idea consists in introducing a new representation of the original dataset where we can control the sensitivity of the count query and at the same time preserve the frequent patterns. This transformation process is based on the concept of *fingerprint* which is extensively applied in string matchings [70, 59]. In our paper, we define fingerprint as follows.

Definition 3.11 (Fingerprint)**.** *Given a set of patterns $F = \{p_1, p_2, \ldots, p_n\}$, for a string s we call the vector $fp_F(s)$ the fingerprint of s on F. Where the i-th component ($fp_F(s)[i]$) represents the number of occurrence of p_i in s.*

Intuitively, the fingerprint of a string represents the contribution of the string on the occurrences of the patterns in F. Furthermore, multiple strings can have the same fingerprints. Our idea is to represent the original strings using their fingerprints on the top-k' patterns. In particular, given a truncated maximum length l'_{max}, for strings of length smaller than l'_{max} we keep their fingerprint directly, otherwise we construct a fingerprint that is as close as possible to the original one and it can be represented with string of length l'_{max}. In this way, we bound the sensitivity of the count query for the patterns on the transformed dataset to l'_{max}, which can be

considerably smaller than the maximum length in the original strings. Clearly, this process may lead to approximation error in representing long strings as illustrated in the Example below.

Example 3.12. *Let $F = \{aa, ab, bb, ac\}$ be a set of patterns, and let $x = aabb$ and $y = bbcacdcbccddaa$ be two strings. Given a maximum length $l'_{max} = 4$ we want to represent these stings with the vectors $\bar{x}$ and $\bar{y}$ respectively. First, for the string x we can notice that the constraint on the maximum length is satisfied therefore we use $\bar{x} = fp_F(x) = [1, 1, 1, 0]$. Second, for the string y the length constraint is violated, so we represent y with a vector $\bar{y} = [0, 0, 1, 1]$ which is a fingerprint of a string of length l'_{max} and has minimum distance from $fp_F(y) = [1, 0, 1, 1]$. Overall, with this representation we lose one occurrence of the pattern aa in the string y.*

As Example 3.12 pointed out, the transformation process may introduce an approximation error in representing long strings since some of the occurrences of the patterns are lost. Therefore, it is important to reduce this error. Here we first introduce some definitions, and then we formalize this problem.

Definition 3.13 (Dominated String)**.** *Given a set of patterns F and two strings x and y. We say that y is dominated by x, denoted as $y \leq x$, if and only if $fp_F(y)[i] \leq fp_F(x)[i]$ for $i = 0, \ldots, |F| - 1$. Furthermore we denote by $\mathcal{D}(x)$ the set of strings dominated by x.*

Therefore, the transformation problem can be formalized as below.

Problem 3.14 (Optimal Constrained Fingerprint)**.** *Given a maximum length l'_{max}, a set of patterns F, and an input string x with fingerprint $fp_F(x)$, find the constrained fingerprint vector $\bar{x}^* = fp_F(x^*)$, where x^* is defined below.*

$$x^* = \arg \min_{y \in \mathcal{D}(x)} \|fp_F(y) - fp_F(x)\|$$
$$\textit{s. t. } |x^*| = l'_{max}$$

In the rest of section, we investigate the challenges for designing an efficient and effective transformation algorithm to reduce the approximation error, and a simple process for determining a suitable value for k' and l'_{max}.

Fingerprint Construction

In this section we propose a heuristic strategy which decomposes the input string into blocks with an associated profit.

Definition 3.15 (Block profit)**.** *Given a block b and a set of patterns F, the profit of b is defined as the ratio between the number of occurrences in b of the candidates in F, over the length of b.*

When a string is received in input, we will return a vector by computing the fingerprint of the string formed by selecting the most profitable blocks till a maximum length is reached. Our transformation procedure is illustrated in Algorithm 5. We divide the input string x into $|x| - b_{min} + 1$ overlapping blocks of length b_{min}, which is defined as the minimum length of the patterns in F. From line 8 to 10 in Algorithm 5, we scan the blocks and we merge two consecutive blocks if they contain some occurrences of the patterns in F. Then at line 13, the algorithm selects the most profitable blocks (i.e. those with the most contribution on the occurrences of the patterns in F) till the accumulated length reaches l'_{max}. Note that if the length exceeds this threshold, we truncate part of the block. We finally combine the contributions of the blocks selected and return the fingerprint.

Refinement

In the previous section we developed a transformation strategy which maps the original set of strings D into a set $\bar{D}$ of vectors (fingerprints). In this new representation, the count of the occurrences for a pattern p_i in F is computed by summing up the i-th component of all the vectors in the transformed data $\bar{D}$. Specifically,

Algorithm 5 String Transformation Procedure

procedure TRANSFORMATION(x, F, l'_{max})
Input: string x; set of frequent patterns F; maximum length l'_{max}
Output: $\bar{x}$ vector representation

$\bar{x} \leftarrow 0$
if ($|x| \leq l_{max}$) **then**
  $\bar{x} \leftarrow fp_F(x)$
  return $\bar{x}$
end if
Let $\mathcal{B}$ be the set of blocks for x initialized with their profit
for (every block i in $\mathcal{B}$) **do**
  Merge two consecutive non-empty blocks i and $i+1$ into i
end for
Update the contribution of the merged blocks
Sort the blocks in decreasing order according to the profit
$\mathcal{S} \leftarrow$ SELECT BLOCKS($\mathcal{B}, l'_{max}$)
for (every block i in $\mathcal{S}$) **do**
  Update $\bar{x}$ with the contribution of i on the pattern in F
end for
return $\bar{x}$
end procedure

given the set of candidate patterns F and the transformed dataset $\bar{D}$, we will perform the query $q = [p_1, p_2, \ldots, p_{k'}]$ on $\bar{D}$, where $p_i \in F$, for $i = 1, 2, \ldots, k'$. Since these vectors represent fingerprints of strings whose lengths are bounded by l'_{max}, it follows from Lemma 3.4 that it is sufficient to perturb the answer of q by adding Laplace noise with parameter ϵ_2/l'_{max} to achieve ϵ_2-differential privacy for the query q on $\bar{D}$. We use these noisy counts to identify the top-k patterns from the set F.

In Section 3.4.2, we will explain the privacy implication of this mechanism with respect to the original dataset D, in particular we show that our transformation and refinement steps guarantee ϵ_2-differential privacy also for the original datset.

Parameter Selection

In this section, we investigate how to select the parameters used in our two-phase algorithm. We defer the study of the impact of ϵ_1 to the experiments section.

Choosing k'. In our first phase, we use our prefix mining algorithms to mine a set of candidate patterns. It is crucial in this step to choose a suitable value of k' so that with high probability the real top-k patterns are contained in the candidate set. Here, we propose a simple way to choose a suitable value to k'. We first make some assumptions on the distribution of the patterns. In particular, we assume that the distribution of the frequency of the patterns follows a Zipf's distribution $f(k; s, N)$, where s is a parameter related to the specific dataset. This assumption is motivated by the fact that in many real scenarios the frequency of the pattern follows a power-law distribution (e.g. frequency of words in English). Furthermore we assume that we can estimate a maximum relative error ΔF for the real frequency of the patterns reported in the first phase. Then, we can find a k' such that the frequency $f_{k'}$ (i.e. frequency of the k'-th pattern) is at most $f_k(1 - \Delta F)$. Intuitively, this means that even in the worst case, that is the real frequencies of the top-k patterns decrease by a quantity ΔF, the real top-k will be still present in the top-k' patterns. Therefore, we can show that it is sufficient to choose a value of k' that satisfies the following inequality.

$$f_k(1 - \Delta F) \geq f_{k'} \tag{3.10}$$

Then normalizing by the sum of all the frequency we obtain:

$$\frac{f_k}{\sum_{i=1}^{N} f_i}(1 - \Delta F) \geq \frac{f_{k'}}{\sum_{i=1}^{N} f_i} \tag{3.11}$$

Using the assumption of the Zipf's distribution (here we assume $s = 1$ for simplicity), we can rewrite the previous inequality as follows.

$$\frac{1}{k}(1 - \Delta f) \geq \frac{1}{k'} \tag{3.12}$$

Solving the inequality above, we obtain that $k' \geq \frac{k}{1-\Delta F} = k(1+\gamma)$, with $\gamma = \frac{\Delta F}{1-\Delta F}$. Unfortunately, the value of ΔF is not easy to estimate, therefore for simplicity in our approach we set $\gamma = 0.5$.

Choosing l'_{max}. In our two-phase algorithm both the parameters k' and l'_{max} play an important role in the final utility. With larger values of k' it is more likely that the top-k patterns are contained in the candidate set. On the other hand, a smaller value of l'_{max} reduces the amount of noise injected on the final count, but it may introduce a larger approximation error since some occurrences of the candidate patterns may not be captured in the transformation. Although we saw how to compute a value of k', the choice of the optimal value for l'_{max} is still very challenging. Therefore, we determine the maximal length l'_{max} in a heuristic way by setting $l'_{max} = \min\{l^*, L\}$. The length l^* is a value that can be computed from the data, for example in our simulation we choose l^* such that the percentage of the strings with length no greater than l^* is at least 85%. The parameter L instead represents an upper bound on the length which determines a maximum value of the error introduced by the noise in computing the final counts.

3.4 Analysis

3.4.1 Complexity Analysis

Prefix Tree phase. The PT-Sample and PT-Dynamic algorithm have running time proportional to the number of nodes in the prefix tree $\mathcal{T}$. First, we can notice that on the level i, we have at most all the possible prefixes of length i defined on the alphabet Σ. Hence, the total number of nodes at level i is bounded by $O(|\Sigma|^i)$. This bound is quite loose, in fact the number of nodes can be much smaller since some prefixes are not extended due to the fact that their counts do not pass the threshold value θ. Moreover, each node performs a counting query on its partition that requires a linear scan. Therefore each level i requires $O(N|\Sigma|^i)$ operations,

where N is the size of the database in input. Since, in the tree there are at most h_{max} levels, the overall running time for our algorithms is $O(N|\Sigma|^{h_{max}+1})$. After the tree is constructed, the consistency constraints require $O(h_{max}^2 N)$ operations as shown in [21]. Finally, the running time for traversing the prefix tree is linear with the number of internal nodes in the tree, hence the overall complexity of our algorithms is $O(N|\Sigma|^{h_{max}+1})$.

Transformation and Refinement. The computational complexity for the Algorithm 5 plays an important role on the overall performance since it is applied on each string in the dataset. Therefore, we reduced its running time as follows. First, for each position i in the input string we keep the set of patterns of F that have an occurrence in i. This can be done in linear time with the length of the string in input when a proper index structure (e.g. prefix tree) is employed for the patterns in F. Second, for each block we store its fingerprint so that the final vector is computed by summing all the fingerprints of the blocks selected. Let $l = |x|$ denote the length of the string x in input, and let $|F|$ be the size of the set F. Since the number of blocks in the input string is at most $O(l)$, then lines 7 to 11 in Algorithm 5 require linear time, while sorting the blocks requires $O(l \ln l)$. The selection of the blocks at line 13 requires $O(l)$ time since the blocks have to be scanned, while the final construction of the vector at line 14 requires $O(l|F|)$. Therefore the final complexity for the Algorithm 5 is $O(l(|F| + \ln l))$. Given N the size of the dataset in input, and l_{max} the maximum length of the strings in input, the complexity for the overall transformation step is $O(N l_{max}(|F| + \ln l_{max}))$. The refinement step requires only a linear scan on the transformed data.

3.4.2 Privacy Analysis

Given the privacy parameters ϵ_1 and ϵ_2, we first show that our two phases achieve ϵ_1 and ϵ_2 differential privacy respectively. Second, we prove that our overall solution satisfies $(\epsilon_1 + \epsilon_2)$-differential privacy.

Theorem 3.16 (Prefix Tree Miner ϵ_1-privacy)**.** *PT-Sample and PT-Dynamic algorithm satisfy ϵ_1-differential privacy.*

Proof. (Sketch) The fact that our PT-Sample and PT-Dynamic satisfy ϵ_1-differential privacy follows by the fact that the counting queries for constructing the prefix tree are issued on disjoint partitions. So from the Theorem 2.5, it is sufficient to guarantee that on any root-to-leaf path in the tree the maxim privacy budget allocated is at most ϵ_1, this follows directly from Algorithm 4. □

For the second phase we have the following result.

Theorem 3.17 (Transformation & Refinement ϵ_2-privacy)**.** *The transformation and refinement steps satisfy ϵ_2-differential privacy.*

Proof. (Sketch) We saw in Section 3.3.3 that the refinement step satisfies ϵ_2-differential privacy with respect to the transformed dataset $\bar{D}$. Therefore, what is left to show is that the transformation process preserves the differential privacy. Intuitively, the set of patterns F used in this step satisfies differential privacy so it does not disclose information about the individual record, however the procedure accesses the original string data directly. It turns out that as long as the transformation is **local**, that is the output only depends on the individual input string, then applying a ϵ_2-differential privacy mechanism on the transformed dataset $\bar{D}$ it also guarantees ϵ_2-differential privacy for the original dataset D. This result has been shown in [79]. Hence our second phase satisfies ϵ_2-differential privacy. □

Therefore, using the sequential composition property in Theorem 2.4 on this sequence of privacy mechanisms, it follows that our overall mining approach satisfies $(\epsilon_1 + \epsilon_2)$-differential privacy.

Theorem 3.18 (Two-phase ϵ-privacy)**.** *Let ϵ_1 and ϵ_2 be the privacy budgets for the first phase and second phase respectively, with $\epsilon_1 + \epsilon_2 \leq \epsilon$. Then, the two-phase algorithm satisfies ϵ-privacy.*

Dataset	size	$\|\Sigma\|$	l_{max}	l_{avg}
MSNBC	989,818	17	14975	4.7
house_power	40,691	21	50	50

Table 3.1: Datasets characteristics

3.5 Experiments

We evaluate the performances of our mining algorithms for two types of patterns: prefix and substring. Furthermore, we investigate the dependency of specific parameters (privacy level in the first phase ϵ_1, and depth of the prefix) on our final results. Finally, we compare our approaches with the n-gram method proposed in [22], which to the best of our knowledge represents the most related work and the state-of-the-art for mining sequential patterns with differential privacy.

Data. In the experiment section we test our approaches on two real sequential datasets. The first is the MSNBC datset, where each string represents a sequence of web pages visited by users within 24 hours on the msn.com domain. The second is the house_power dataset which is derived from a numeric dataset representing the energy power consumptions of individual household over 47 months. The original data appears as time-series, therefore for our sequential data we first discretized these values and we successively construct a string record from every 50 samples. A summary of the two datasets is reported in Table 3.1. Both the MSNBC and the household energy consumption datasets are available at the UCI machine learning repository [1].

General Settings. Our two-phase algorithms are denoted by PT-S* and PT-D*, where as the first phase we employ the sample based (Section 3.3.1) and the dynamic model tree (Section 3.3.1) respectively. If not specified in the text the parameters for the algorithms assume the default values reported in Table 3.2. The utility measure in these experiments is the F_1 score, which is a combination of

[1] http://archive.ics.uci.edu/ml/datasets

Parameter	Description	value
ϵ	Total privacy parameter	0.1
ϵ_1	Privacy parameter in the first phase	0.85ϵ
I	Range of variable lengths	[2,7]
h_{max}	Depth of the prefix tree	10
k	Number of frequent patterns to mine	60
$\|S\|$	Sample size for PT-S*	10% of $\|D\|$

Table 3.2: Parameter Description

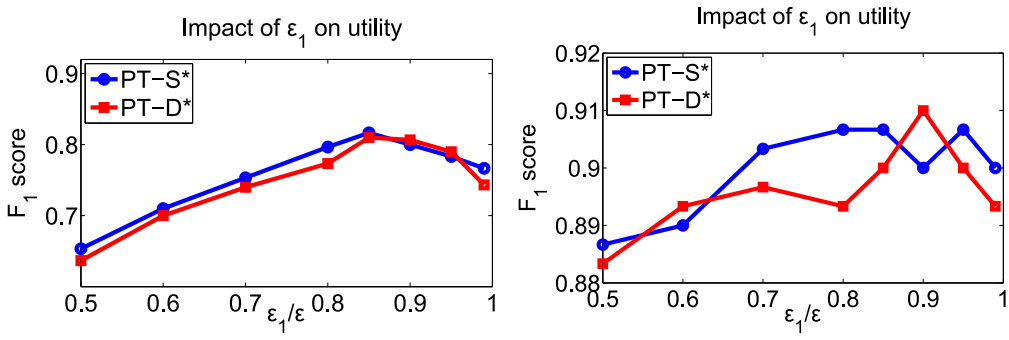
precision and recall for the mined top-k patterns.

3.5.1 Impact of the parameters on the utility

First of all, we start to study the impact of our specific parameters on the final utility of the mined results.

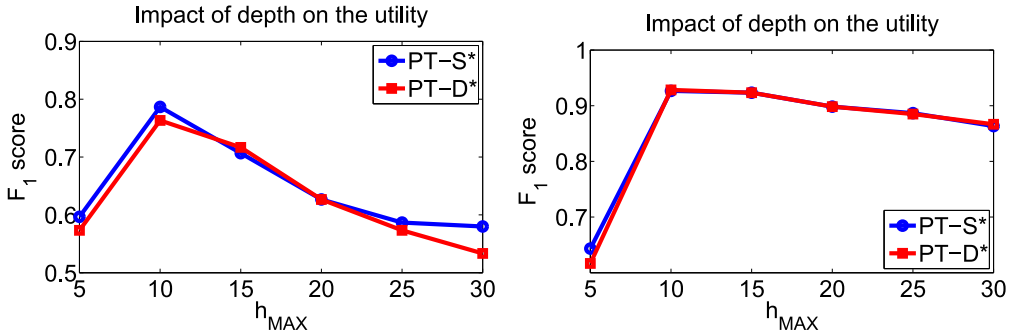
Allocation of the privacy budget ϵ_1. The impact of the privacy budget ϵ_1 that we use in the first phase of our algorithms on the final utility is illustrated in Figure 3.3. We can notice that in both datasets increasing ϵ_1 has beneficial impact on the utility; however, after a certain value an increment in this parameter quickly degenerates the results. This phenomena is caused by the tradeoff between the quality of the candidate set that we generate in the first phase and the noise injected in the second phase. Figure 3.3 is an indication of the effectiveness of our transformation, since even using a small privacy budget (15%-20% of total ϵ) in the second phase it does not compromise the final results. Therefore in our experiments, we will use $\epsilon_1 = 0.85\epsilon$.

Depth of the tree h_{max}. Figure 3.4 illustrates the impact of the depth in the prefix tree used in the first phase on the final utility. Ideally, we would like to have a prefix tree as high as possible to capture all the patterns in the data. However, as the height of the tree increases, less privacy budget is available for each node



(a) house_power (b) MSNBC

Figure 3.3: Impact of the value of ϵ_1 on the final utility.



(a) house_power (b) MSNBC

Figure 3.4: Impact of the depth of the prefix tree on the final utility.

which leads to a larger perturbation noise in the counts. This is more evident in the house_power dataset since it is smaller than MSNBC (i.e. smaller counts). In our experiments, we fix $h_{max} = 10$ since with this setting our algorithms well perform on both datasets.

3.5.2 Comparison for mining frequent patterns

In this part of the experiments we evaluate our algorithms in mining patterns against the state-of-the art approach n-grams proposed in [22] and with the prefix

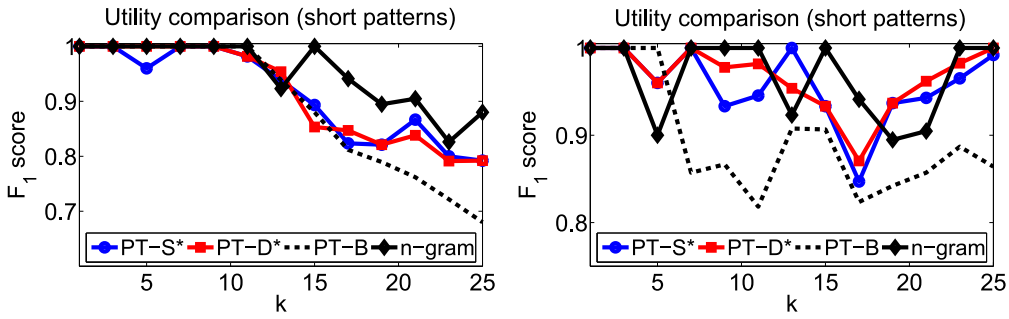
tree base algorithm PT-B (base line) [21, 23] that we briefly described in Section 3.3.1. For the n-grams approach we use the default setting suggested by the authors in [22], while for the PT-B we set a maximum depth of the tree equals to the value of h_{max} in our two-phase algorithm. In our experiments, we consider the mining task both for substring and prefix patterns.

Mining Frequent Substrings

To understand the performance of our mining algorithms for substring patterns, we study several possible scenarios. We first consider the task of mining short frequent patterns with small k values, that is the case where the frequencies of the patterns is more likely to be well separated. In the second case, we study the problem of mining longer patterns with large k values, which could be more challenging since even a small perturbation noise in the frequency could drastically change the top-k patterns. We also investigate the impact of the privacy parameter and the length of the patterns on the final utility.

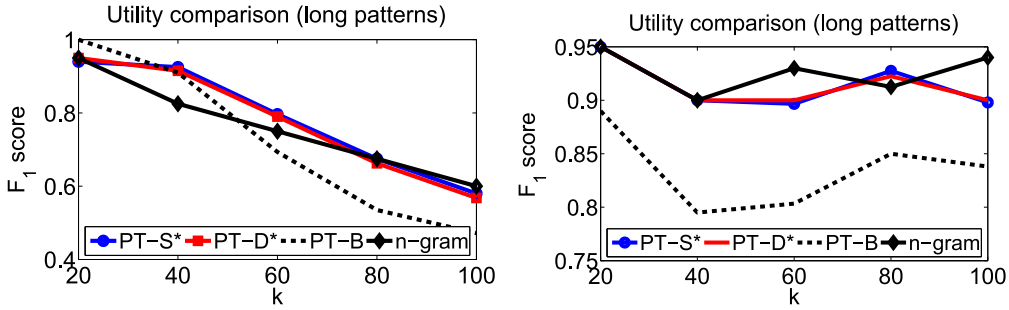
Mining Short Patterns. We consider patterns p of variable length in the range $[2, 3]$ symbols. The results on the two datsets are reported in Figure 3.5. For the `house_power` dataset the approaches have similar performance for $k < 15$. For k in the range $[15, 25]$ all the approaches decrease their performance. The results for PT-B quickly drop, while our solutions are quite stable. In this range of k values, the n-gram approach performs slightly better than our solutions. For the `MSNBC` the results from our algorithms are comparable with those from the n-gram method. Furthermore, we can see that the base line suffers in both datasets.

Mining Long Patterns. Contrary to the previous setting, we now consider longer patterns. In particular we focus on mining patterns of length in the range $[2, 7]$ for k values between 20 and 100. Figure 3.6(a) shows the utility on the `house_power` dataset. We observe that our mining techniques constantly provide higher utility than the n-gram strategy for values of k in between 20 and 80, while for larger



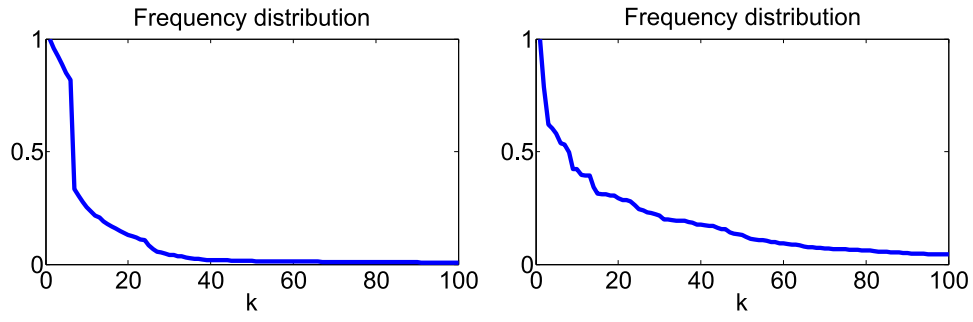
(a) house_power (b) MSNBC

Figure 3.5: Comparison for mining short substring patterns.



(a) house_power (b) MSNBC

Figure 3.6: Comparison for mining long substring patterns.



(a) house_power (b) MSNBC

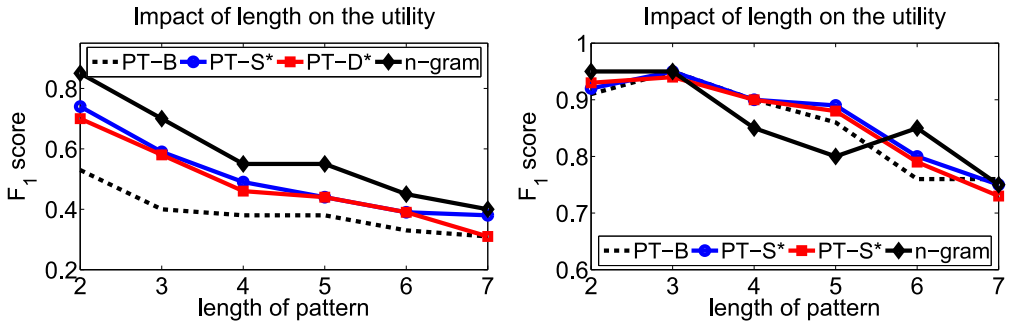
Figure 3.7: Frequency distribution of long substring patterns.

values this gap becomes smaller. The baseline provides comparable results only for small k, then its performance rapidly decreases. The results for the MSNBC dataset are reported in Figure 3.6(b). In this case our strategies closely follow the results from the n-gram miner, while the baseline constantly suffers providing an utility of 10% less than our approaches.

From these mining results we can observe a considerable gap in the utility between the patterns mined from house_power and MSNBC. In fact, we can see from Figure 3.7(a) that the frequency distribution of the patterns in the first dataset rapidly decreases while in web browsing data this behavior is not so extreme, as shown in Figure 3.7(b). This does not provides a good separation between the frequencies of the patterns for large values of k (e.g. $k > 40$) which makes the mining of the patterns in house_power more challenging.

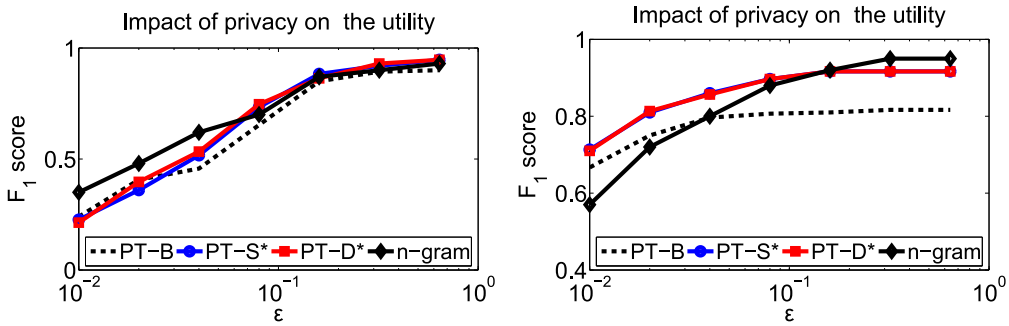
Length of the patterns. In the mining process we consider patterns of variable length, so it is important to understand how the length of the patterns affects the final utility result. Intuitively, long patterns are more difficult to mine for two reasons. First, their absolute value of frequency is lower than short patterns. Second, in general there is not a good separation in term of frequency. Hence, the perturbation noise is more likely to drastically change their frequency and shuffle their order in the top-k. We can observe this phenomena from Figure 3.8, where increasing the length of the patterns mined makes the utility to quickly drop. Our approach well performs against the baseline in both datasets. Compared with the n-gram our two-phase solutions provide slightly lower result in the house_power dataset while in the MSNBC our results are often better.

Total privacy budget ϵ. The impact of the total privacy parameter ϵ on the final utility is reported in Figure 3.9. As the intuition suggests, we can see that increasing the privacy budget (less privacy) increases the utility. In the house_power dataset all the approaches are close, while in the MSNBC there is a distinction. In particular, compared with the n-gram technique our solutions provide better util-



(a) house_power (b) MSNBC

Figure 3.8: Impact of the length of the patterns k=20.



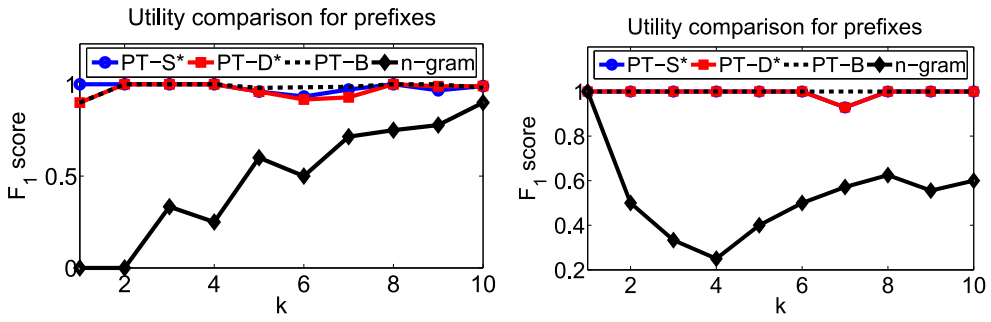
(a) house_power (b) MSNBC

Figure 3.9: Impact of the value of ϵ on the final utility.

ity for small privacy budget, while for $\epsilon > 0.1$ our result follows the one from the n-gram model. Furthermore, in this dataset we can clearly see the advantages of our techniques with respect to the baseline method.

Mining Frequent Prefixes

Since by their nature, the frequency of the prefixes are considerable lower than the frequency of substring patterns, it is more reasonable to focus the mining task on short prefixes. In this setting, we examine the performance of the approaches for prefix mining and report the results for both datasets in Figure 3.10. For both



(a) `house_power` (b) `MSNBC`

Figure 3.10: Comparison for mining prefix patterns, I=[2,3].

the datasets, the n-gram approach provides quite poor utility. The reason is that for this approach the information about the prefix patterns is not preserved. In fact, only substrings are considered in the construction of the n-gram model, and therefore the prefixes cannot be retrieved. On the other hand, in our approaches the use of the first phase allows us to capture these frequent prefix patterns which leads to very accurate results. In a similar way, the baseline PT-B based on the prefix tree structure also achieves good utility in this setting.

3.6 Conclusion

In this paper, we proposed a novel techniques for the differentially private mining of frequent sequential patterns. First, we developed two prefix tree mining techniques that make use of statistical information to carefully calibrate the perturbation noise. Second, we introduced a local transformation technique on the original string records which reduces the amount of noise injected by the Laplace Mechanism. We showed, how to combine these two key strategies into a novel two-phase mining algorithm. A set of extensive experiments proved that our solutions provide comparable results with the state-of-the-art [22] in mining frequent substrings, while achieving significantly superior results for mining frequent pre-

fixes at the same time. Future research directions include the mining of noisy frequent patterns (e.g. with error and gap), as well as the mining of patterns over streams.

Chapter 4

Privacy Preserving Record Linkage

4.1 Preliminaries

In this section, we introduce some notations and definitions related to our approach. First, we briefly present notions concerning string records and the concept of embedding. Second, we review the model of differential privacy which is used as our privacy model.

4.1.1 Basic Definitions

Let x be a sequence of n symbols defined as follows $x = x_0x_1 \cdots x_{n-1}$ where each x_i is defined over a finite alphabet Σ. We denote the length of the string x with $|x|$. Furthermore, given a string x, we represent a substring from position i to j in x with $x[i,j] = x_ix_{i+1} \cdots x_j$, where $0 \leq i \leq j \leq n-1$. For the rest of the paper, we also refer to $x[i,j]$ as a *gram* of length $j-i+1$. A common similarity measure between strings is the Edit distance d_{Edit}, known as Levenshtein distance, which given two strings it measures the number of edit operations needed to transform one of the string in the other.

Definition 4.1 (Edit Distance [50])**.** *The Edit distance between two strings x and*

y is defined as the minimum number of character edit operations necessary to transform x into y. A single character edit operation can either replace, delete or insert a character in x or in y. We denote the Edit distance between x and y by $d_{Edit}(x, y)$.

Using this definition of the similarity metric, we denote the metric space by $(\Sigma^*; d_{Edit})$ as the pair: space of all possible strings, and Edit distance. We informally introduce the notion of embedding used in the paper as a map $\rho : \Sigma^* \rightarrow \mathbb{R}^k$, where k is the dimensionality of the embedded space. This transformation maps the string records into vectors in $\mathbb{R}^k$. The distance in this new space is computed by using Euclidean distance which is denoted by d'.

Given the Edit distance as similarity measure between strings, we state the problem of record linkage that we consider in this paper as follows.

Problem 4.2 (Record Linkage)**.** *Given two sets D_A and D_B of string records and a maximum distance t, find $\mathcal{M} \subset D_A \times D_B$, such that $\mathcal{M} = \{(x, y)$ s. t. $d_{Edit}(x, y) \leq t\}$ (matching records) and no information about the individual records $(x, y) \notin \mathcal{M}$ (non-matching records) is disclosed.*

In the rest of the paper, we will refer to the case where the threshold $t = 0$ as *exact match*, while the case $t > 0$ as *approximate match* since it is sufficient for the records to be within t edit operations to be considered as a match.

4.2 Overview of Proposed Solution

As we described in the previous section, secure transformation techniques map original records in a new space where they can be securely matched. Typically, this transformation process is data independent so that the data owners can share the information required to map the records and obtain consistent matching results in the new space. For example, in Scannapieco et al. [64], the transformation is performed using a reference set of random strings, called base, that data holders

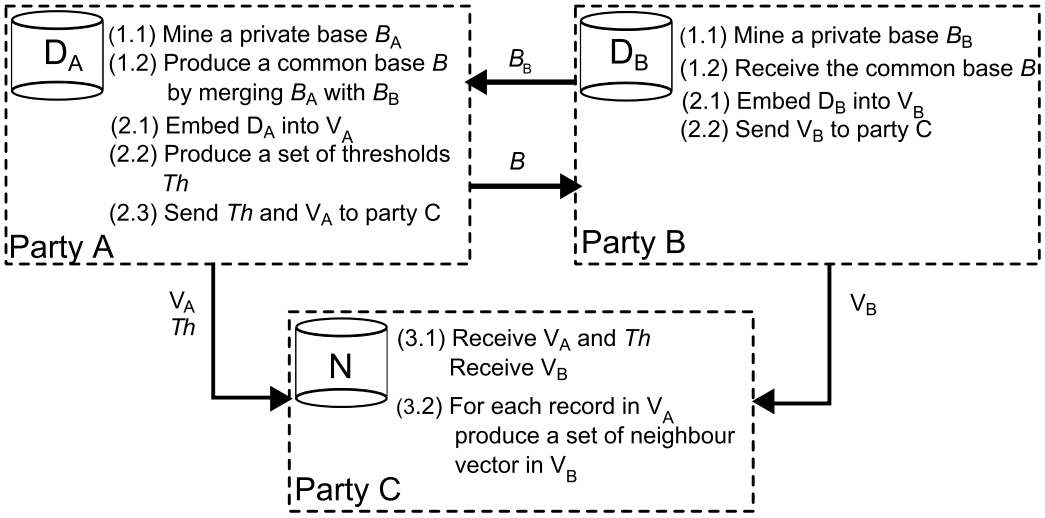
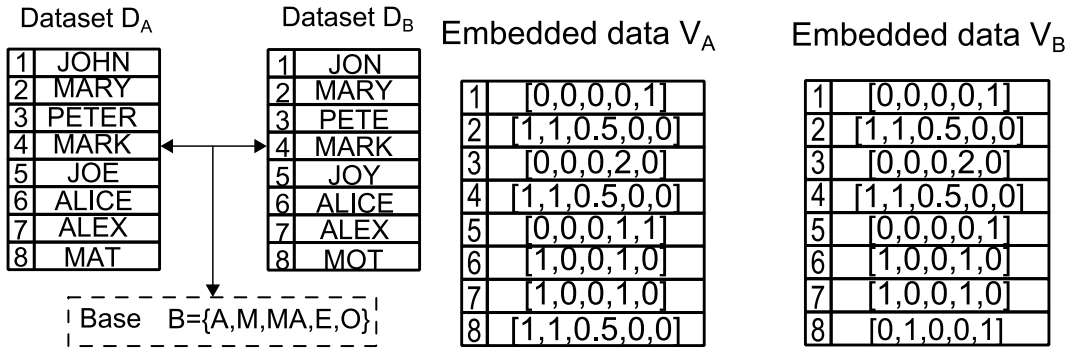


Figure 4.1: Overview of the Secure Protocol.

share during the process. Although the randomly generated base does not disclose information about the original records, it may lead to poor utility results due to its generality.

In our solution, we propose an embedding technique that uses a set of *grams* (i.e. substrings) extracted from the original data as a base to map the original records into vectors. In particular, our base is a set of frequent variable length grams that are mined from the original data. A gram of length q (q-gram in short) is a substring $x_0x_1 \cdots x_{q-1}$ of the original records. We show that this definition of base allows our map to better represent the records compared to a random base and it leads to better utility in the matching phase. Since this base is shared among the data holders, to avoid privacy leakage about individual record, the frequent grams are mined with guarantee of differential privacy. In this way, we bound the adversary inference ability of determining the presence of any individual record in the original data by observing the base.

Our strategy requires the presence of a third party (not necessary trusted) denoted by C whose task consists in matching the records in the embedded space.



(a) **Base Generation** (b) **Embedded Data**

Figure 4.2: Example of Mining and Embedding of the data.

The matching is performed in the vector space using the Euclidean distance d' as a similarity measure. We notice that this step is common to all the secure transformations and in principle any secure protocol can be employed to provide additional security in matching the transformed vectors.

Our overall protocol is illustrated in Figure 4.1. The generation of the frequent grams is performed by the data holders in the **mining phase**. First, each data holder privately mines a set of frequent grams and then a common base is defined. Then each party embeds its data using the common base and the resulting vectors are sent to the third party. We denote this step as **embedding phase**. Finally, the third party matches the transformed records in the **matching phase**. A summary of the steps is listed as follows.

1. **Mining Phase:** Parties A and B apply a differentially private algorithm to mine their respective databases D_A and D_B and compute *private* bases $\mathcal{B}_A$ and $\mathcal{B}_B$. One of the two parties is in charge of merging the two bases and it produces a shared base $\mathcal{B}$ of frequent variable length grams. Figure 4.2(a) illustrated the mining and base generation steps. Parties A and B mine their respective dataset and they define a common base $\mathcal{B}$ formed by the following grams $\{$`A,M,MA,E,O`$\}$.

2. **Embedding Phase:** Each party A and B, by using the shared base generates a set of vectors V_A and V_B respectively, representing the strings in the original datasets. An example of embedded records using the base $\mathcal{B}$ is illustrated in Figure 4.2(b), where each component in the vectors is computed as the number of occurrences of each gram in the record normalized by the length of the gram. For example, for the string "JOHN", the resulting vector is $[0, 0, 0, 0, 1]$, where only the component associated with the gram "O" is non-zero. These vectors are sent to the third party C which is in charge of the matching. Given a maximum value of edit distance t in the original space, the party A generates a set of threshold values $Th = \{th_1, th_2, \ldots, th_N\}$. These values represent a distance threshold in the embedded space for each string s_i, $i = 1, 2, \ldots, N$ in D_A to use in the matching phase. In particular, each th is a personalized threshold for the string $s \in D_A$ and it represents the maximum distance between $\bar{s}$ and $\bar{z}$, such that the string z is close to s (i.e. within t edits). This set of thresholds is sent to C.

3. **Matching Phase:** The third party C, for each vector $\bar{s} \in V_A$ and its threshold value $th \in Th$ returns a set of neighbor vectors $\mathcal{N}(\bar{s})$ computed as follows:

$$\mathcal{N}(\bar{s}) := \{\bar{z} \in V_B \text{ s. t. } d'(\bar{s}, \bar{z}) \leq th\} \tag{4.1}$$

where the distance between vectors is computed using the Euclidean similarity metric. For example, by using threshold value $th = 0$ (exact match), the record "JOHN" in D_A is matched with "JON" and "JOY" in D_B, since the Euclidean distance between their respective vectors is 0.

4.3 Mining Phase

In this section, we describe the steps required to generate the base for our embedding technique. The base has important implications both for the utility and privacy of our transformation. First, the set of grams in the base determines how well records will be represented in the new space and therefore it impacts the final utility. Second, since the base is shared among data holders it is necessary that no information about individual record can be inferred from observing the base.

In our strategy, we select a base for the embedding from the original data. This choice is motivated by several reasons. First, in record linkage scenarios, it is reasonable to assume that records being matched/linked have similar properties (e.g. same alphabet). Therefore, by constructing a base from the original records we can incorporate this information in the embedding step. On the other hand, a base randomly generated is defined in a generic way and therefore cannot well reflect changes in different datasets. Second, our base construction solely depends on the original data and does not require a direct user intervention. In this way, we liberate users from the challenging task of selecting and tuning a base for the transformation, making the entire process more easy and robust.

Our base is composed by *frequent* grams directly mined from the original databases. Formally, given a positive integer k, a minimum length q_{min} and a maximum length q_{max}, we mine the top-k frequent q-grams where $q \in [q_{min}, q_{max}]$ and we use this set as a base for the embedding. In this way, we obtain a base that is a good representative for all the strings in the databases and intuitively well preserve the structure of the original data in the vector space. A similar idea has been successfully applied in many approximate string matching problems [75, 51, 70], where strings are matched using the partial information captured by their substrings. However, a direct use of the mined grams from a party in the protocol may disclose information about individual records. For example, a malicious user could infer the presence of a sensitive string record by observing the grams in the

base. To address this problem, we propose a mining algorithm that generates a base satisfying the rigorous notion of differential privacy. Hence, the base can be shared without incurring the risk of disclosing information about individual records.

For mining the frequent grams we employ our prefix tree mining approach that we described in Chapter 3. In principle, other differentially mining techniques can be used as a black box for computing the frequent grams such as [10, 22, 14]. For example, we could use our recent two-phase algorithm [10] to retrieve the frequent grams. Although this new approach represents the state-of-the-art of differentially private sequential pattern mining and it could greatly improve the quality of the mined grams, we may not observe drastically changes in the final matching results. In fact, as we will point out in the experiment section, the entire process is quite robust against changes in the base. Therefore, later we will describe only our prefix-tree based approach for mining the set of frequent grams.

4.3.1 Base Generation

The grams computed in the mining phase satisfy differential privacy, that is, they do not reveal information about the presence or absence of any individual user's record in the original dataset. In our protocol, we use this important fact to enable data holders to share the information about their mined grams and construct a common base $\mathcal{B}$ for the embedding. One can argue that a base from public domain knowledge could be used as alternative to our approach. Although, sharing such a base does not cast privacy treats on individual records, this choice could compromise the final utility since the selected grams may not provide a good representation of the records in the datasets.

In our approach, the construction of the base $\mathcal{B}$ is performed by merging the frequent grams from the data holders. In principle, this step is not trivial since there are many possible ways to define a common base. In our solution, we construct

the base $\mathcal{B}$ by selecting the top-k grams from the two sets of top-k frequent grams mined both from datasets. We use this simple strategy because we believe that in real settings the original datasets required to be linked have similar structures (e.g. records with similar content) which likely will lead to close distributions of the frequent grams. In future, we plan to use different strategies to select the common base, for example by selecting the most diversifying grams or using a weight for the grams associated with their respective data owner.

4.4 Embedding Phase

In this section, we describe the embedding strategy for mapping string records into vectors using the base of frequent grams. Furthermore, we analyze the impact of the grams on the final results by studying the error introduced by mistakenly omitting or selecting grams in the base.

4.4.1 Embedding

Given the base as a set of substrings (grams) $\mathcal{B} = \{g_1, g_2, \ldots, g_k\}$, the embedding function maps each string into a vector using the occurrences of grams in the base. In particular, let $Occ_s(g)$ denote the set of positions in s where a gram g occurs,

$$Occ_s(g) := \{i \in [0, |s| - |g|] : s[i, i + |g| - 1] = g\} \tag{4.2}$$

and let $O_s(g) = |Occ_s(g)|$ be the number of occurrences of a gram g in a string s. Given these notions, we define our embedding function as follows.

Definition 4.3 (Embedding Function)**.** *Let ρ be a function defined as $\rho : \Sigma^* \to \mathbb{R}^k$, that maps a string s into a real vector $\bar{s}$. Given the base $\mathcal{B} = \{g_1, g_2, \ldots, g_k\}$, each coordinate of the vector $\bar{s}$ is defined as: $\bar{s}[i] = O_s(g_i)/|g_i|$, for $i = 1, \ldots, k$.*

The definition above is very general and the set of grams $\mathcal{B}$ could be defined in different ways. As we mentioned early, in our approach, we use the set of frequent

grams as a base for our embedding. It turns out that with this choice, the distance between records is quite well preserved in the embedding as we will show in the experiments in Section 4.7. The use of frequent grams for the base is motivated by the fact that these substrings are frequent, and therefore they are shared among many strings which allows us to represent them in the embedded space. Furthermore, the fact that these grams have variable length allows the map to capture the strings with a different granularity which is crucial when we consider the approximate match of the strings (i.e. match with some edit operations).

After the embedding step is performed, for each original string record a real vector is generated using the common base. In this vector space, we use the Euclidean distance d' as similarity function between vectors. Then, for any pair of transformed records $\bar{s}_A \in V_A$ and $\bar{s}_B \in V_B$, the pair (s_A, s_B) is a match if $d'(\bar{s}_A, \bar{s}_B) \leq th$.

4.4.2 Impact of the Grams

The frequent grams are used in constructing the base for the embedding and determining the components in the vector representation of the original strings. However, due to the privacy requirements, our mining approach perturbs the real frequency of the grams which could impact the final utility. Therefore, to help users in deciding the privacy parameter ϵ and the size of the base k, it is important to understand the impact of the grams on the final results. In this section, we analyze the impact of the grams on the map by considering two possible cases: (1) frequent grams are mistakenly omitted from the base, and (2) infrequent grams are mistakenly included in the base.

Intuitively, we can observe that for case (2), the matching process is more tolerant to the presence of infrequent grams in the base. In fact, due to their low frequency, these grams will produce zero components in majority of the vectors and their presence may only affect the matching results of few records. On the

other hand, in case (1), the omission of some frequent grams from the base has a greater impact on the matching results. In fact, a frequent gram is likely to be shared among many records and therefore omitting it from the base will lead to a loss of non-zero components for many vectors, potentially increasing the mismatched records. In the follows, we study the impact of both cases and we will show that only with small probability we may eventually miss a frequent gram from the mined base with our prefix tree mining algorithm.

Error Estimation.. In our analysis, we consider a simplified case where only one gram g is mistakenly included or omitted from the base and we relate its error to the number of records affected by g. Therefore this quantity, denoted with the variable $n(g)$, represents an upper bound on the number of matches that we could possibly lose with the gram g. We can assume that the grams follow some distributions (e.g. Zipf) and our goal is to compute the expected number of records affected by g, hence $E[n(g)]$. Assuming that all the grams have the same length $|g|$, the following lemma provides an upper bound for $E[n(g)]$.

Lemma 4.4 (Error Upper Bound)**.** *Let l be the length of the records and n be the total number of records, then given a gram g the quantity $E[n(g)]$ can be upper bounded by $n \cdot p_g \cdot l$, where p_g is a probability value depending on the frequency distribution of the grams.*

Proof. Let $r_i(g)$ denote an indicator random variable that represents if the i-th record r_i contains the gram g, formally:

$$r_i(g) = \begin{cases} 1 & \text{if } g \text{ appears in the } i\text{-th record} \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

$$\tag{4.4}$$

Then, we have that: $n(g) = \sum_{i=1}^{n} r_i(g)$. We can estimate $Pr[r_i(g)]$ as follows:

$$Pr[r_i(g)] \leq 1 - Pr[g \text{ does not occur in } r_i] \tag{4.5}$$

In modeling the probability distribution of a gram g to appear in a record r_i, we assume independency between the grams and the position in the records. In particular, we consider $Pr[r_i[j, j+|g|-1] = g] = p_g$ for every $j = 1, \ldots, l-|g|+1$, where p_g represents the probability for the gram g to appear in any position of r_i. Hence, we have that $Pr[g \text{ does not occur in } r_i] = (1-p_g)^{l-|g|+1}$, which leads to

$$Pr[r_i(g)] \leq 1 - (1-p_g)^{l-|g|+1} \tag{4.6}$$

Then using the fact that $(1-x)^y \geq 1 - x \cdot y$ for $y \geq 1$ and $0 \leq x \leq 1$, we have that:

$$E[n(g)] = \sum_{i=1}^{n} E[r_i(g)] \leq n \cdot p_g \cdot (l - |g| + 1) \leq n \cdot p_g \cdot l \tag{4.7}$$

□

Assuming that the grams follow a Zipf's distribution, we can approximate p_g as $p_g \approx \frac{f_g}{\sum_{i=1}^{G} f_i}$, where f_g represents the frequency of g and G denotes the total number of grams. Therefore, the expected number of records which contains the gram g can be upper bounded by $(n \cdot f_g \cdot l)/(\sum_{i=1}^{G} f_i)$. As we expected, frequent grams have a higher impact since more vectors may contain them compared to infrequent grams.

Error Probability.. Now we quantify the probability of mistakenly omitting or including a gram in our base. We recall that our mining algorithm computes the frequency of the grams combining the frequency of multiple prefixes. In particular, let $\tilde{f}_g$ be the frequency released by the prefix tree for the gram g, and let f_g denote its real frequency. Furthermore, let $\mathcal{P}(g)$ be the set of nodes in the tree having g as a suffix, then we can write $\tilde{f}_g$ as follows.

$$\tilde{f}_g = \sum_{\nu \in \mathcal{P}(g)} \tilde{c}(\nu) = \sum_{\nu \in \mathcal{P}(g)} c(\nu) + \sum_{\nu \in \mathcal{P}(g)} Lap(1/\epsilon_\nu) \tag{4.8}$$

In the construction of the tree, some prefixes are not extended due to the fact that their noisy counts do not pass the threshold value θ. Therefore, the sum of the

counts for the prefixes in the tree containing g as a suffix can be upper bounded by the frequency of g in the real data f_g as: $\tilde{f}_g \leq f_g + \sum_{\nu \in \mathcal{P}(g)} Lap(1/\epsilon_\nu)$. We denote the sum of these Laplace noises with the random variable $Y = \sum_{\nu \in \mathcal{P}(g)} Lap(1/\epsilon_\nu) = \sum_{i=1}^{n} Lap(1/\epsilon_i)$. Therefore, the probability of missing a frequent gram g from the base can be represented by $Pr[Y \geq (f_g - f_k)]$, while the probability of inserting a infrequent gram is $Pr[Y \geq (f_k - f_g)]$, where f_k denotes the frequency of the k-th gram in the base.

Lemma 4.5 (Error Probability)**.** *The probability of mistakenly omitting or selecting a gram g of frequency f_g from the base, using our prefix tree algorithm is $Pr[Y \geq |f_g - f_k|] \leq exp\{-\frac{(f_g - f_k)^2}{8\gamma^2}\}$, where $\gamma \geq \max\{\sqrt{\frac{\lambda}{2 \cdot \epsilon_{min}}}, \sqrt{\sum_{i=1}^{n} 1/\epsilon_i^2}\}$, and $\lambda < \epsilon_{min}/2$.*

Proof. We start by considering the generating function of Y, then for any λ we have that:

$$\begin{aligned} Pr[Y \geq \alpha] =& Pr[e^{\lambda \cdot Y} \geq e^{\lambda \cdot \alpha}] \leq \\ \leq& \frac{E[e^{\lambda \cdot Y}]}{e^{\lambda \cdot \alpha}} = \frac{\prod_{i=1}^{n} E[e^{\lambda \cdot Lap(1/\epsilon_i)}]}{e^{\lambda \cdot \alpha}} \end{aligned} \tag{4.9}$$

Then, for $\lambda < \epsilon$, the generating function for the Laplace random variable $Lap(1/\epsilon_i)$ is $E[e^{\lambda \cdot Lap(1/\epsilon_i)}] = 1/(1 - \frac{\lambda^2}{\epsilon_i^2})$. Furthermore, using the inequality $(1 - x)^{-1} \leq 1 + 2x \leq e^{2x}$ for $0 \leq x < 1/2$ and assuming $\lambda < \sqrt{\epsilon_i/2}$, we have

$$E[e^{\lambda \cdot Lap(1/\epsilon_i)}] = 1/(1 - \frac{\lambda^2}{\epsilon_i^2}) \leq e^{2\frac{\lambda^2}{\epsilon_i^2}} \tag{4.10}$$

Then, we impose $\lambda < \sqrt{\epsilon_{min}/2}$, and by using the previous result in equation (4.9), we have :

$$\begin{aligned} \frac{\prod_{i=1}^{n} E[e^{\lambda \cdot Lap(1/\epsilon_i)}]}{e^{\lambda \cdot \alpha}} \leq& \frac{\prod_{i=1}^{n} e^{2\frac{\lambda^2}{\epsilon_i^2}}}{e^{\lambda \cdot \alpha}} \\ =& e^{-\lambda \cdot \alpha + 2\lambda^2 \sum_{i=1}^{n} 1/\epsilon_i^2} \end{aligned} \tag{4.11}$$

Then setting $\lambda = \frac{\alpha}{4\gamma}$, and $\gamma \geq \max\{\sqrt{\frac{\lambda}{2 \cdot \epsilon_{min}}}, \sqrt{\sum_{i=1}^{n} 1/\epsilon_i^2}\}$, where $\epsilon_{min} = \min\{\epsilon_i\}$, we have

$$Pr[Y \geq \alpha] \leq e^{-\frac{\alpha^2}{8\gamma^2}} \tag{4.12}$$

Plugging in $\alpha = |f_g - f_k|$, we obtain the result desired. □

The result above shows that the probability of mistakenly omitting a frequent gram from the base decreases exponentially with the difference of frequency. On the other hand, the probability of mistakenly inserting some infrequent grams is quite high, since it may be $f_g \approx f_g$.

From this analysis, we can observe that although missing a frequent gram could affect the matching results for many vectors, it is very unlikely that such event occurs.

4.5 Matching Phase

The matching phase is crucial in determining the matching records in the embedding space. In this step a similarity comparison between records is performed using the Euclidean distance metric and according to the distance value between vectors the original records are marked as match or non-match. The distance value has a great impact on the overall linking performance; therefore, it is crucial to compute a threshold value th in the embedded space as tight as possible to the real value.

Below, we describe two possible criteria to compute the threshold value to match the embedded records. The first approach aims to decide a global threshold value that hold for all the pair of records, while the second strategy specialize the threshold for each records pair.

4.5.1 Global Threshold

In this section, we study the problem of computing a global threshold for matching pairs of records. In particular, given an edit distance threshold t in the original space, we want to determine a threshold value th in the embedding space such that no vectors representing true matches are discarded. Furthermore, we want th to be as tight as possible in order to reduce the number of false positives. To understand how to compute such a threshold value, we start by considering a simple scenario where the grams have the same length q and the base contains all the possible grams. In this setting, we can see that for any pair of strings s_1 and s_2 such that $d_{Edit}(s_1, s_2) \leq t$ (i.e. matching records), the number of shared grams between these two records are at least

$$\max\{|s_1|, |s_2|\} - q + 1 - t \cdot q, \tag{4.13}$$

where the term $\max\{|s_1|, |s_2|\} - q + 1$ denotes the number of shared grams of length q and $t \cdot q$ represents an upper bound on the number of grams they could lost with t edits.

Unfortunately, in our case we can not directly apply the bound from equation (4.13) to compute th, for two reasons. First, the number of shared grams between each record pair depends on the base, and second the grams in the base have variable length. To obtain a value of th in our case, we directly consider the vector representation of the strings. In this representation, the number of non-zero components represent the grams shared between records and the base. Furthermore, the embedding procedure incorporates the gram length information in the vector representation by normalizing the contribution of each grams by their length. Using these two facts, the following lemma provides an upper bound for the distance in the embedded space.

Lemma 4.6. *For the proposed embedding function ρ that maps strings into vectors, there exists a constant $\alpha = \sqrt{k}$ such that for any pair of strings s_1, s_2 in the*

original space the following inequality holds:

$$d'(\rho(s_1), \rho(s_2)) \leq \alpha \cdot d_{Edit}(s_1, s_2) \tag{4.14}$$

Proof. Consider the Euclidean distance between vectors embedded using a base with k grams, as follow:

$$d'(\rho(s_1), \rho(s_2)) = d'(\bar{s}_1, \bar{s}_2) = \sqrt{\sum_{i=1}^{k} (\bar{s}_1[i] - \bar{s}_2[i])^2} \tag{4.15}$$

Furthermore, let $d_{Edit}(s_1, s_2) = t$ be original distance. Given the following two observations: (1) t edits can affect at most $t \cdot q$ grams of length q, (2) the contribution for the grams in the embedding is scaled by their length, then for each coordinate i following relationship holds:

$$(\bar{s}_1[i] - \bar{s}_2[i])^2 \leq t^2 \tag{4.16}$$

Then we have that $d'(\bar{s}_1, \bar{s}_2) \leq \sqrt{k} \cdot t$, hence $d'(\rho(s_1), \rho(s_2)) \leq \alpha \cdot d_{Edit}(s_1, s_2)$. □

The lemma above shows two important results. First, the embedding is contractive by scaling down the Euclidean distance by $1/\sqrt{k}$. Second, larger base size may not help to improve the matching results since k contributes to increase the distortion. Despite the importance of such result, this upper bound could be too loose leading to poor utility in practice. In principle, both parties can pursue the task of finding a global threshold value by inspecting each pair of records, as follows:

$$th = \min_{d} \{d \geq d'(\bar{x}, \bar{y}) | d_{Edit}(x, y) \leq t\}, \quad \forall x, y \tag{4.17}$$

However, solving Equation (4.17) require both parties to test all possible pairs of records to find the optimal global threshold value. Clearly, both from privacy and computational reasons, this task cannot be directly performed in the protocol. Therefore, often it turns out that th is left as parameter to be chosen by user. In our experiment section, we test our approach by tuning different values of th to maximize the utility.

4.5.2 Personalized Threshold

As we have seen from the previous section, the use of a global threshold for matching records based on the analysis could lead to a loose threshold. Furthermore, from the experiment section, we will see that finding the optimal threshold value is hard without a priori knowledge, and even a slight change in the threshold value can lead to a considerable loss of accuracy on the final result. To address these limitations, we propose to compute a personalized threshold value for each string. The design of this strategy is motivated by the following reasons. First, each string shares a different number of grams with the base. Hence for those strings that have a large number of shared grams, a personalized threshold can better represent the original distance. Second, by computing a threshold for each string, we overcome the problem of estimating a threshold suitable for all strings which we have shown to be difficult.

For each string s and the set $\mathcal{N}$ of neighboring strings within t edit distance, the personalized threshold is the maximum distance between the embedded vector of s and the set of embedded vectors of $\mathcal{N}$. To compute the personalized threshold we adopt the algorithm proposed in [75] which originally has been introduced to perform approximate string matching. In our case, we modify this approach so that we can keep track of the impact of each gram on the final distance in the embedded space.

Given a string s, we start to initialize two data structures as follows:

$$D[i] = \sum_{g \in \mathcal{B}} \left(\frac{\delta_s(i, g)}{|g|} \right)^2, \quad i = 0, 1, \ldots, |s| - 1 \tag{4.18}$$

$$\delta_s(i, g) := \begin{cases} 1 & \text{if } g \text{ overlaps } s \text{ at position } i \\ 0 & \text{otherwise} \end{cases} \tag{4.19}$$

$$P[i] = \max_{g \in \mathcal{B}} \{ j < i - |g| - 1 \text{ where } j \in Occ_s(g) \} \tag{4.20}$$

For each position i in s, the entry $D[i]$ represents the total contribution that grams overlapping at position i may have in the coordinates of $\bar{s}$ if some edit operations

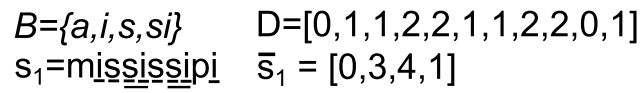


Figure 4.3: Impact of edit operations on the embedded vector.

occur at position i. On the other hand, $P[i]$ denotes the nearest position $j < i$ where a gram of the base occurs in s with no overlap on i.

Example 4.7. *Consider the original string $s_1 = mississipi$ and a base $\mathcal{B} = \{a, i, s, si\}$ as illustrated in Figure 4.3. Then the embedded version of s_1 is $\bar{s}_1 = [0, 4, 4, 1]$, since both "i" and "s" occurs four times in s_1 and have length 1, while "si" occurs twice and it has length 2. Furthermore, the vector D has 10 coordinates, where the largest values are at positions $3, 4$ and $6, 7$ representing the overlap of "i" and "s" with "si". Therefore, with 2 edits, the maximum distance for a vector $\bar{s}_2$ representing a matching string s_2 is $\|\bar{s}_1\|_2 - \sqrt{2+2} = \sqrt{33} - 2$.*

Using these structures and given a maximum number of allowed edit operations t, we are interested in computing the largest allowable distance th in the embedded space for each string s. This value is computed by filling a $t \times |s|$ matrix $\mathbf{T}$, where the entry $\mathbf{T}[i, j]$ denotes the maximum change in the components of the vector $\bar{s}$ when i edit operations are allowed on the first j positions of s. Algorithm 6 illustrates how the matrix $\mathbf{T}$ is updated according to the presence of edit operations at each position j in s. The personalized threshold value is then computed by taking the square root of $\mathbf{T}[t, |s| - 1]$.

Theorem 4.8 (Personalized Th Correctness)**.** *Algorithm 6 computes the maximum distance in the embedded space where a string within t operations from s can be located.*

Proof. We need to show that line 4 of Algorithm 6 gives the correct maximum contribution $\mathbf{T}[i, j]$. Consider the case when at position j in the string s no edit operations occur. Then the contribution remains the same as in the previous position $\mathbf{T}[i, j - 1]$. Similarly, when multiple edit operations occur at position j, we

Algorithm 6 Dynamic Algorithm for computing th

```
1: procedure PERSONALIZED TH(s, D[·], P[·], t)
   Input: String s; D[·], Pr[·], and edit distance t.
   Output: Threshold value th

2:     for (i = 0, 1, ..., t) do
3:         for (j = 0, 1, ..., |s| − 1) do
4:             T[i, j] = max{T[i, j − 1], T[i − 1, P[j]] + D[j]}
5:         end for
6:     end for
7:     return th = √T[t, |s| − 1]
8: end procedure
```

have to add the contribution from grams overlapping in j ($D[j]$) to the maximum contribution of having $i-1$ edit operations in the position before j ($\mathbf{T}[i-1, P[j]]$). This concludes the proof. □

Complexity Analysis. The overall computation cost can be divided into two components: setup and threshold computation. The setup phase requires to initialize the data structures D and P. This step does not depend on t so it can be done once on the entire dataset (e.g. off-line preprocessing). On the other hand, the computation of the personalized threshold depend on t and it is performed each time the approximate match for a string is required. For this step the computation cost is determined by Algorithm 6 which has running time $O(t \cdot |s|)$.

4.6 Security Analysis

In this section, we analyze the security of our record linkage protocol. In particular, we want to show that no information regarding non-matching records is disclosed to any adversary participating in the protocol.

4.6.1 Adversary Model

The adversary model that we adopt in our analysis is based on the definition of *semi-honest* adversary (known also as *Honest-But-Curious Adversary* (HBC)) [37, 17], which forces the adversary to obey to the prescribed actions in the protocol but it does not prevent the adversary to learn extra information from the protocol transcript. This means that the adversary can perform several attacks including dictionary and frequency based attacks to infer the non-matching records.

In our protocol, the information that is exposed to each party is different and varies with the role of the party in the protocol. It will be useful in our analysis to summarize this information as follows. (1) *Data holder parties (Party A and B)*: the only information available from the protocol is the base $\mathcal{B}$ and the matching records returned at the end of the matching process. Furthermore, each party knows only his/her own original and embedded data. (2) *Third party (Party C)*: the only information received by this party is the thresholds produced by A, and the set of embedded records received by the data holders. Using this model, we analyze the security of our protocol in two situations. We first consider the presence of one adversary which plays the role of one party and second we study the security in the case of collusion.

4.6.2 Security against one adversary

We start considering the case where the adversary plays the role of one party in the protocol. Two different scenarios are possible under this assumption: (i) one of the data holders (A or B) is an adversary, (ii) the third party C is an adversary. In case (i), the only information shared among the two parties A and B is the common base $\mathcal{B}$ which is proven to satisfy differential privacy. Therefore, the adversary cannot learn anything about the presence or absence of any individual record in the dataset held by the other party. In case (ii), the adversary can see only the embedded data from the other two parties in the protocol and the set of

thresholds, but it has no access to the base, nor the original data of either party. At this point the adversary could mount some attacks to reverse the embedding, for example a frequency attack to infer the base. However, if no priori knowledge on the characteristic of base (e.g. length of the grams) or domain information about the records embedded (i.e. alphabet, length of the records) is available, it is infeasible for the adversary to learn about the original non matching records. Although the threshold values are available to the third party, this information reveals only the maximum distance between matching records and it does not leak additional information about the original data. In fact, the adversary cannot fully take advantage of the threshold values since the embedding base is not available to the third party.

It is important to notice that the differential privacy mechanism employed in generating the base is crucial in providing privacy and security when one of the data holder is an adversary. In fact, since the base of grams is differentially private, the privacy is guaranteed between the two data holders. On the other hand, our protocol does not extend this guarantee to the third party since both embedding and threshold are data-dependent.

4.6.3 Security against collusion

In the case that one of the party and the third party collude, analyzing the potential disclosure risk becomes extremely challenging. In fact, the adversary has a larger amount of information that it can use to break the security. For example, without loss of generality, we can assume that parties C and A are malicious and they share their information in order to identify the non-matching records of B. In this case the adversary knows: the shared base $\mathcal{B}$, the embedded data V_A, V_B, the original data D_A, and the matching records of B in the embedded space. With this information, the adversary can easily determine which vector in V_B represents a non-matching record, and then he can try to identify the original string records

that map to non-matching vectors by performing several attacks. Clearly, in this setting it is not possible to provide formal and complete guarantees of security. However, we simulated several dictionary attacks and it turns out that for every non-matching vector, the adversary could obtain a large number of candidate strings that map to it. As a consequence, the adversary is unlikely to learn about the original non-matching records. We will show in the experiments in Section 4.7.4 that our approach is resilient with respect to different dictionary attacks.

This resistance to dictionary attacks is largely due to our data-dependent embedding strategy. Dictionary attacks have been shown to break the security of the traditional protocols based on hashing functions, since the attacker can learn the hashing map. However, in our protocol, the embedding function is data-dependent and therefore a brute force attack using random strings will have few overlaps with the base, and hence a large number of strings will be mapped to the same vector. As we will show in Section 4.7.4, this data-dependent mapping enhances the utility for matching records in the original datasets (low number of collisions) and, at the same time, prevents the disclosure of information when an adversary tries to learn the original data by embedding other random data (high number of collisions). This property gives an important value to our protocol compared to other approaches based on traditional hashing functions.

To add an additional secure layer to our approach, we can replace the party C with a SMC protocol, where the computation of the distance between embedded vectors can be performed using a secure inner-product protocol as in [**?**]. In fact, the distance between vectors $\bar{x}$ and $\bar{y}$ can be approximated as follows: $d'(\bar{x},\bar{y})^2 = \|\bar{x}-\bar{y}\|_2^2 = \|\bar{x}\|_2^2+\|\bar{y}\|_2^2-2\,\langle\bar{x},\bar{y}\rangle$, where only the inner-product $\langle\bar{x},\bar{y}\rangle$ is computed in a secure way by the SMC protocol.

	NAMES	CITIES
N	150k	5k
l_{max}	15	23
l_{min}	4	3
l_{avg}	7	8

Table 4.1: Datasets

4.7 Experiments

In this section, we present a set of experiments aiming to evaluate and understand the overall utility of our protocol.

In our experiments, we use two real datasets, NAMES[1] and CITIES[2]. The NAMES dataset, contains a list of the most frequent surnames from the Census 2000, while CITIES lists the top 5000 most populated cities in U.S. in 2008. Some of the statistics of the datasets are summarized in Table 4.1, where l_{max}, l_{min} and l_{avg} are the maximal, minimal and average lengths of the strings, respectively. The experiment and algorithm parameters, if not specified in the descriptions, assume the default values as reported in Table 6.2. The protocols were implemented in Java, and the simulations were conducted on an Intel Core i5 2.5Ghz PC with 4GB of RAM. For mining and linking utility, we report the standard F_1 metric based on precision and recall.

4.7.1 Embedding Performance

We start investigating some of the properties of our proposed embedding. In particular, we show the benefits of using a base formed of frequent grams over random grams with two type of experiments.

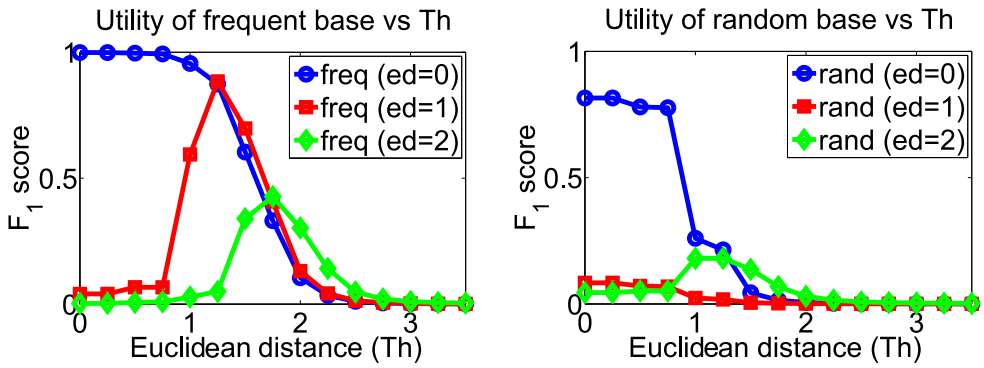
First, we measure the utility results in terms of F_1 score for matching the records

[1] http://www.census.gov/genealogy/www/data/2000surnames/

[2] http://www.census.gov/popest/cities/SUB-EST2008-4.html

Symbol	Description	Value
ϵ	Privacy par.	0.1
k	Base size	75
q_{min}	Min gram length	1
q_{max}	Max gram length	3
t	Edit operations	$\{0, 1, 2\}$
h_{MAX}	Depth of PT	l_{avg}
θ	Threshold	as in [21]

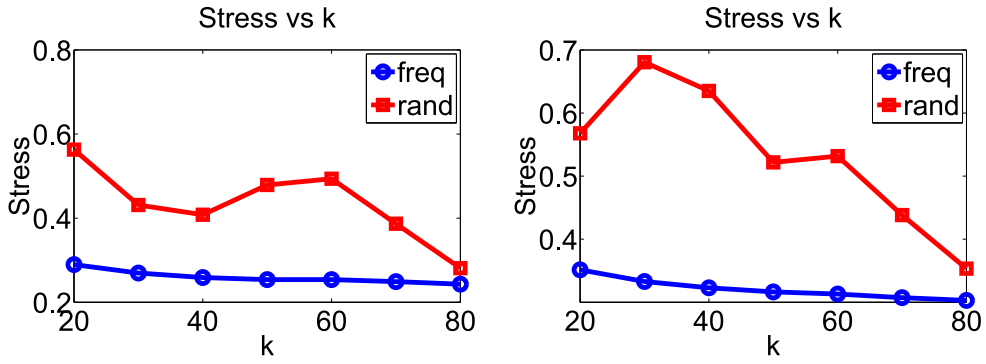
Table 4.2: Parameters



(a) Frequent grams (b) Random grams

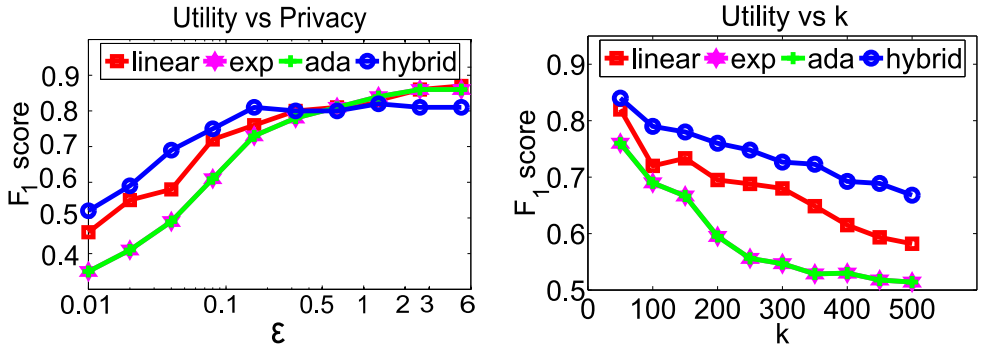
Figure 4.4: Impact of grams base on the utility.

in the embedding space, where the transformation uses a base of frequent grams and a base formed by random substrings respectively. The results for this comparison on the NAMES dataset are reported in Figure 4.4(a), 4.4(b). Similar results have been obtained for the CITIES dataset and we omit them here. From these figures, the base formed with frequent grams leads to higher utility in matching the transformed records than a random base. Furthermore, the use of frequent grams allows our map to better preserve the structure of the original data, and the embedding results are considerably more robust in the presence of edit operations.



(a) NAMES dataset (b) CITIES dataset

Figure 4.5: Quality measure for the embedding.



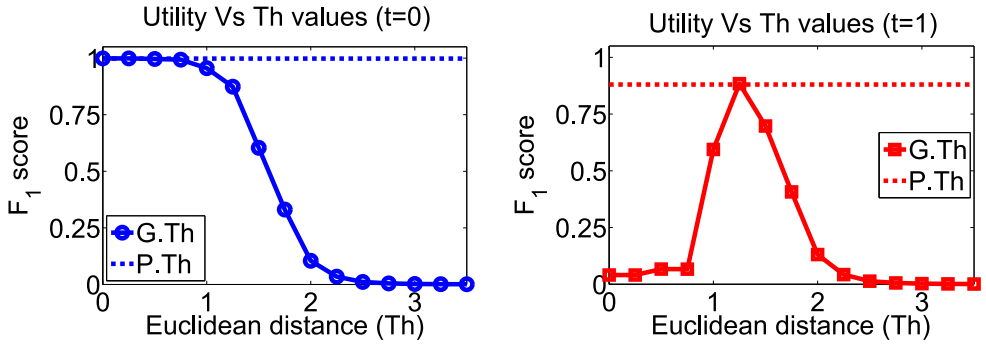
(a) Utility vs Privacy (b) Utility vs Dim.

Figure 4.6: Mining Performance.

Second, we study how the distance between records is preserved in the embedding. In particular, we consider a random sample S of records from both datasets in input and we measure the overall deviation in distance caused by the embedding. As a measure we use the notion of *stress* [**?**], which is defined as follows:

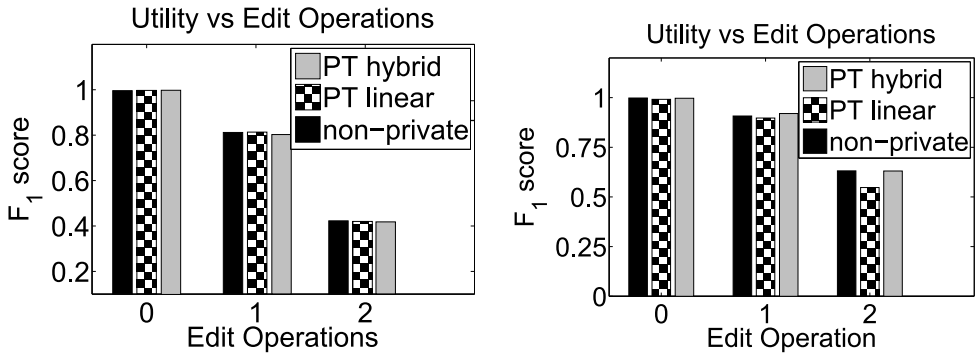
$$stress(S) = \frac{\sum_{x,y \in S} (d_{Edit}(x,y) - d'(\bar{x},\bar{y}))^2}{\sum_{x,y \in S} d_{Edit}(x,y)^2} \tag{4.21}$$

From Equation (4.21), small values of stress indicate the ability of the embedding to preserve the relative distance between the records after being mapped. We



(a) Exact Match (b) Approximate Match

Figure 4.7: Matching Performance.



(a) Utility in NAMES (b) Utility in CITIES

Figure 4.8: Utility vs distance threshold.

measure the stress produced by our strategy using a frequent and a random base respectively. In our simulation, we use a sample S of 1k records from entire dataset and we report the stress by performing multiple runs, and taking the average of the measured stress for different size of the base (k). The results are illustrated in Figure 4.5(a), 4.5(b). As the dimensionality increases the stress for both bases decreases. This is not surprising since larger values of k enable the embedding to provide a finer representation of the records. Furthermore, the stress for the base of frequent grams is considerably lower than random grams which indicates

a lower deviation in the distance.

4.7.2 Mining Performance

In this set of experiments, we examine the utility of the mined frequent grams with respect to the privacy parameter ϵ and the value k. We evaluate the performance of our prefix-tree algorithm (PT) with different privacy allocation strategies. We denote by linear, hybrid, ada and expo the linear, hybrid, adaptive and exponential allocation strategies proposed in Section 4.3. Here, we report only the performance obtained with the `NAMES` dataset, since similar results have been observed on the `CITIES` dataset as well.

In Figure 4.6(a), as the privacy budget increases (less privacy), the utility for all the techniques increases. This is intuitive, since larger values of ϵ lead to less noise in perturbing the counts, hence better grams frequency are obtained. Among the various allocation strategies, hybrid and linear achieve the best result. For small values of ϵ, the hybrid allocation strategy and ada are more accurate than the linear approach, since the privacy budget can be better allocated among the level of the tree. On the other hand, for larger values the linear has better utility.

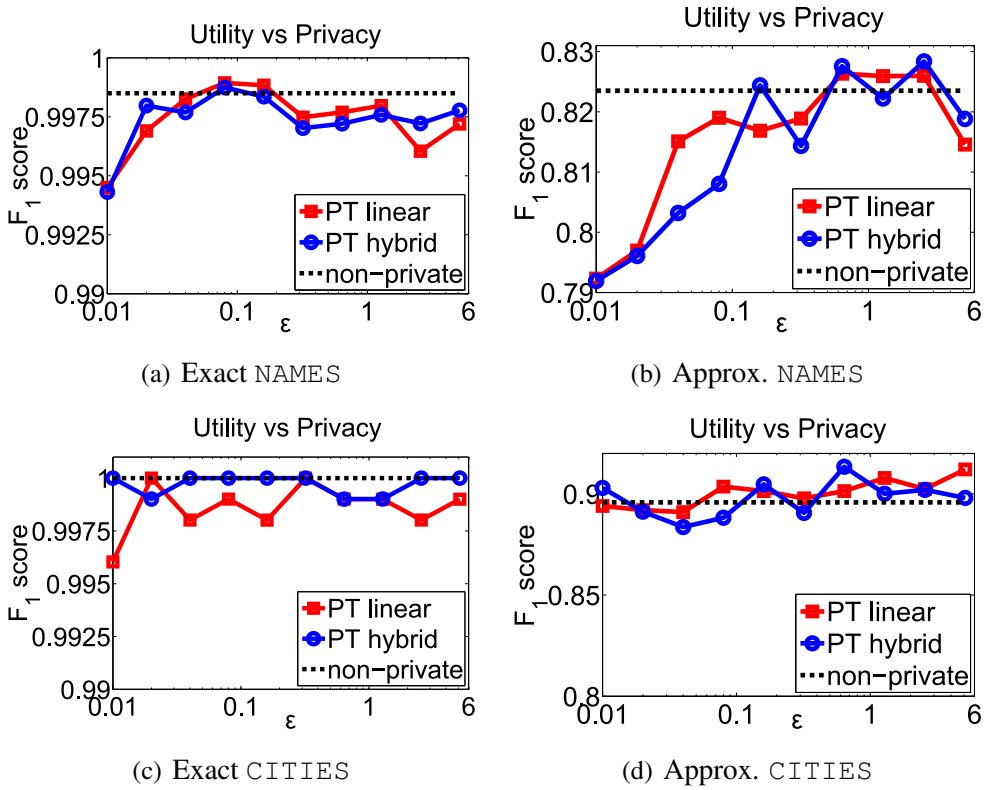
Figure 4.6(b) confirms the intuition that increasing the number of grams k decreases the utility for the mining results. In fact, for larger values of k, it is more likely to report infrequent grams as frequent because the relative frequencies of the grams are closer while the noise injected remain the same. This increases the chances to shuffle the grams, hence report grams that are infrequent as frequent. In this case, the use of the hybrid strategy is more robust with the dimensionality k.

In light of these results, we will equip our PT algorithm with hybrid and linear allocation strategies, denoted as PT-hybrid and PT-linear, respectively. These approaches will be used in the mining step for the comparison with the state-of-the-art technique for the rest of the experiments.

4.7.3 Linking Performance

To evaluate the performance of our solutions, we consider three different settings: (1) the matching criteria (i.e. impact of global threshold vs. personalized threshold in the embedding space), (2) the edit distance (i.e impact of the threshold in the original space), and (3) the privacy level (i.e. impact of the base in the mining phase). The details of this evaluation are reported in the rest of the section. For computational reasons, for the NAMES dataset, the utility is measured on sets of sampled records, which required a considerably lower running time to perform pair-wise comparison with respect to the entire set of records.

Impact of the matching criteria.. We examine the results between the matching based on global and personalized threshold. To demonstrate the advantage of our personalized threshold, we compare the utility results in the record linkage obtained both in case of exact match and approximate matching (one edit operation allowed) on the NAMES dataset. The results for the exact matching are reported in Figure 4.7(a), while the results with one edit are illustrated in Figure 4.7(b). In our experiments, we test different values of global threshold in order to find the best value that achieves the highest utility. The personalized threshold approach (P. Th) achieves a F_1 score as high as the best result produced by the global threshold strategy (G. Th). This clearly shows the advantage of personalized threshold approach, as the optimal global threshold is hard to determine without a priori knowledge in practice. In fact, even if we use the upper bound from Section 4.5.1, the range of possible th values is from 0 to 75 while the best value is achieved when th is between 1 and 2. Furthermore, the global threshold approach is very sensitive to threshold values where even small changes can lead to a considerable loss of utility in the matching results. This phenomena is clear in Figure 4.7(b), where the ability to provide useful matching results for the global threshold strategy is limited to one value. For the rest of the experiments, we will use a matching criterion based on the personalized threshold which has been shown to be very ef-



(a) Exact NAMES (b) Approx. NAMES

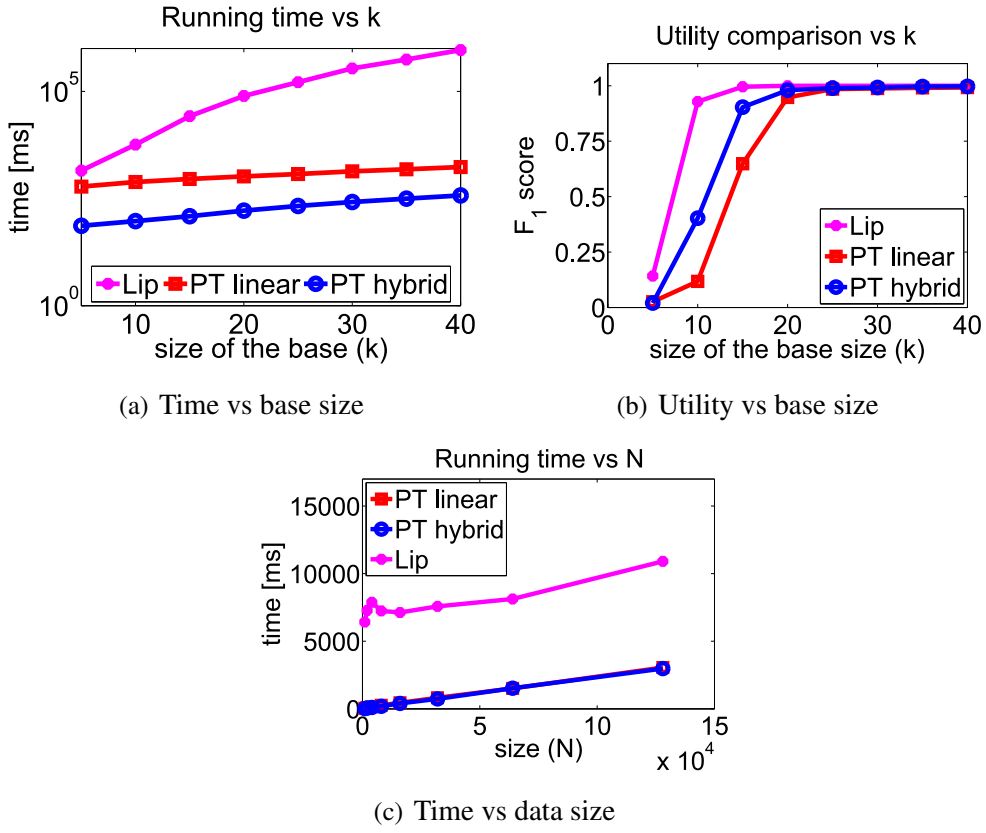
(c) Exact CITIES (d) Approx. CITIES

Figure 4.9: Utility evaluations.

fective and more robust than the global threshold based method.

Impact of edit operations.. In Figure 4.8(a), 4.8(b), we present the impact of edit operations on the utility of our protocol. In our matching criteria we consider as matching records those strings that are within t edit operations. As the edits increase the overall utility decreases for all the strategies tested. This is not surprising, since as the number of edits increase, more grams are affected leading to fewer grams shared with the base, hence lower utility results.

The edit operations considered in these simulations vary from 0 (exact match) to 2 edits (35% of the avg length of the strings in the dataset). The F_1 score decreases from 0.99 in the case of exact matching to 0.36 when 2 edits are allowed for the



(a) Time vs base size

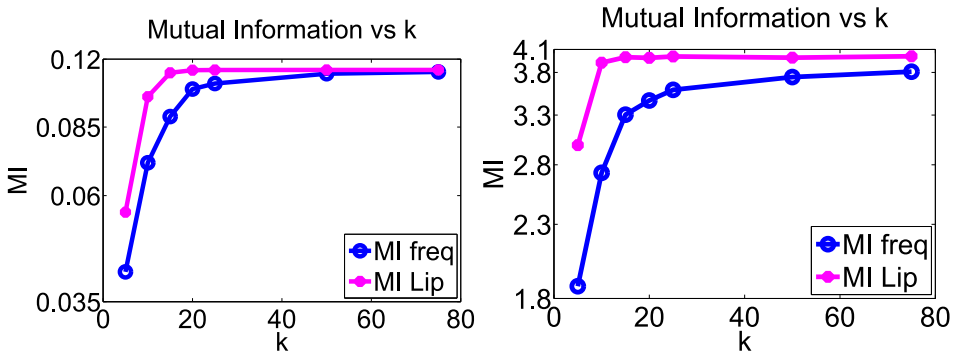
(b) Utility vs base size

(c) Time vs data size

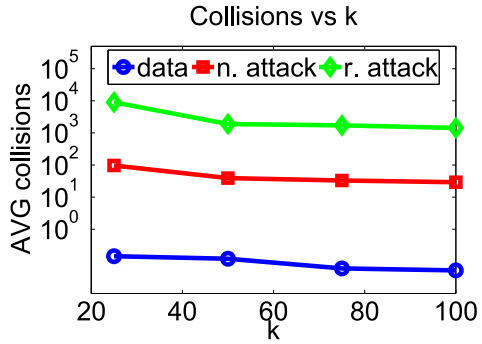
Figure 4.10: Performance Comparison.

NAMES dataset, while for the CITIES dataset stops at 0.60. As mentioned earlier, this phenomena is due to the fact that the presence of edits drastically decreases the number of shared grams between the matching strings and hence increases the Euclidean distance between their corresponding vectors. Second, in the datasets considered, many strings are short and therefore even the presence of few edits could drastically change them. Concerning the performances, the PT hybrid and linear approaches closely follow the utility results provided by the non private miner, which again emphasizes the centrality of the embedding step on the final utility.

Impact of the privacy parameter.. The relationship between the privacy param-



(a) MI for NAMES (b) MI for CITIES



(c) Number of collisions

Figure 4.11: Security Evaluation

eter ϵ and the final utility of our protocol is reported in Figure 4.9(a) - 4.9(d). To understand the impact of the differentially private mining algorithms employed in the mining phase, we compare our complete solution with a protocol where the mining phase is done in an exact way using a non-private miner. As we can see, the utility of our solutions approach the results obtained with the non-private algorithm as ϵ increases. First, these experiments show the efficacy of our protocol, indeed we obtain F_1 scores higher than 0.99 for the exact matching in both datasets. When approximate matching is considered, the utility moderately drops to 0.82 in the NAMES dataset, and to 0.9 in the CITIES dataset. Second, these results point out that the utility gap in the mining phase has minimal effect on the

final utility. This fact suggests that the embedding map plays a crucial role in the overall utility.

Performance Comparison.. We compare our approach with the state-of-the-art technique for secure record linkage proposed in [64]. The running time with respect to the size of the input data are reported in Figure 4.10(c) while the dependency with respect to the dimensionality of the embedding space are shown in Figure 4.10(a). In our approach, we measure the time needed to mine the base for each party, to generate the shared base, and to embed the data. While for the approach proposed in [64], we measure the time required to generate the base using the heuristic and produce the embedding map.

In Figure 4.10(c), the running time for all the approaches scales linearly with the size of the data in input. However, for our techniques, the time needed is considerably lower. Also in Figure 4.10(a), the performance of our techniques are significantly superior. Finally, in Figure 4.10(b), we report the utility with different base sizes on the `CITIES` dataset. In this case, the Lipschitz embedding provides slightly better utility for smaller bases, while our strategies achieve similar results when $k \geq 15$. However, the dependency of the running time with respect to k is exponential for [64], while in our approach is considerably lower.

4.7.4 Security

In this section, we evaluate the security for our protocol. First, we assume that the third party is an adversary that wants to infer the original data from the embedded vectors without knowing the base for the embedding. Second, we consider a collusion between the parties A and C, which allows the adversary to access the base.

Non-collusive attack setting.. In this case, we use the *mutual information entropy* (MI) to measure the information that the adversary can gain by seeing the embedded records in his/her task of inferring the original strings. The mutual

information is a standard measure for evaluating the security in terms of the dependency of the encoded text with the original text [**?**]. Recently, Durham et al. [28] used this measure for evaluating the security of several record linkage techniques. Let X and $\bar{X}$ be two random variables representing the original records and the embedded vectors respectively. The mutual information $MI(X, \bar{X})$ between these two variables is expressed in term of entropy H as:

$$MI(X, \bar{X}) = \sum_{x \in X, \bar{x} \in \bar{X}} Pr[x, \bar{x}] \log_2 \frac{Pr[x, \bar{x}]}{Pr[x] Pr[\bar{x}]} \tag{4.22}$$

In our case, X is defined over a set of 1k strings randomly sampled from the original dataset, while $\bar{X}$ represents the random variable associated with the embedded vectors. We measure the mutual information between the embedded records produced by our frequent base embedding and the approach in [64] with the original strings. We performed multiple runs and took the average of the mutual information values. The results are reported in Figure 4.11(a), 4.11(b). For both datasets, our approach leads to lower mutual information with respect to [64] indicating higher independency between the embedded vectors and the original strings and thus a greater security. For both approaches, the mutual information increases as the dimensionality increases since more information can be captured in the embedding space.

Collusive attack setting.. In this case, the adversary knows the shared base and the non-matching records of B in the embedded space, and therefore he/she can perform a dictionary attack to identify the strings that embed to the non-matching vectors. In our simulations, we estimate the number of possible candidate strings that could embed to the same vector. Clearly, if this number is large it implies that for the adversary it is hard to deterministically identify the original records of B.

We measure the number of collisions by simulating two different attacks: (1) brute-force and (2) dictionary attack, which are based on the level of knowledge the adversary has. In (1), the adversary knows only the length l of the

strings, so this attack consists in embedding all the possible strings of length l (`r. attack`). In (2), the adversary knows the domain of the string records. In our scenario for matching names, the adversary mounts an attack using a set of the most popular names (`n. attack`). We measure the average number of collisions of these attacks, and compare them with the collisions when original data are embedded (`data`). The results are reported in Figure 4.11(c). As we can see, the number of collisions is always greater than 10^3 for `r. attack` and almost 10^2 for `n. attack`. This means that the adversary can infer the original string records of B with very low probability (0.001 and 0.01 respectively for these attacks). We point out that only a small number of collisions is incurred when the original dataset is embedded, which allows our matching strategy to achieve good utility.

4.8 Conclusion

In this chapter, we presented a novel privacy preserving record linkage technique for string records. First, we developed an embedding technique that performs a secure transformation using a differentially private base extracted from the original data. Second, we presented a complete study of the embedding procedure initially proposed in [14] and provided a formal analysis of its properties. Furthermore, we introduced the concept of personalized threshold for matching records in the embedded space. As we have shown from our experiments, this new strategy without requiring any a priori knowledge, allows us to obtain utility results as high as the best results obtained by setting an optimal value of the global threshold. We showed that our overall strategy presents comparable performance with the start-of-the art technique in [64] while guaranteeing formal differential privacy and better scalability. In [15], we deployed our solution into a complete record linkage tool named LinkIT which enable privacy preserving record linkage. As a future research direction, we are interested in integrating secure transformation

techniques for different attribute types.

Chapter 5

Analytics over Data Stream

In this chapter, we illustrate our solutions for privately computing the length of the longest increasing subsequence in a streaming setting.

5.1 Differentially Private Computation of the LIS - A Baseline Approach

In the rest of the chapter, we present our solutions for computing the length of the LIS in the stream. Our approaches require the stream to be *time-bounded*, we assume in fact that the length of the stream is T and it is given a priori.

Here we consider a baseline approach that solves the problem of privately computing the length LIS by perturbing directly its real value at every time point. In particular, for every new element $\sigma(i) = a_i$ in the stream, the algorithm first computes the real $LIS(\sigma[0, i])$ (e.g. using any non-private solution, Patience Sorting in this case) and then it adds a perturbation noise η_i. Given the privacy parameter α, due to the composition property of differential privacy, to obtain an overall mechanism of α-differential privacy, the baseline approach applies the Laplace mechanism at each time point with parameter $\alpha' = \alpha/T$. For each new incoming

element, it samples a Laplace variable $\eta_i \sim Lap(1/\alpha')$ which will be used to perturb the real value of $LIS(\sigma[0,i])$. Therefore, at every time i, the algorithm will answer the LIS query by returning $\tilde{l}(\sigma[0,i]) = LIS(\sigma[0,i]) + \eta_i$. We can observe that the sensitivity for the LIS function is 1, since replacing an element from the stream may change the length of the longest increasing subsequence by at most 1. Therefore, perturbing the real value of $LIS(\sigma[0,i])$ with η_i is sufficient to achieve privacy. The utility of this approach is reported in the following theorem.

Theorem 5.1 (Baseline utility)**.** *The baseline algorithm is* $(\frac{\beta\sqrt{T}}{\alpha} \ln \frac{1}{\delta}, \delta)$*-useful for computing the longest increasing subsequence.*

Proof. The released length of the LIS at each time i is obtained by perturbing the real length of the LIS with Laplace noise. Therefore, at every time step in the stream we have that the additive error from the noise can be bounded as follows:

$$Pr[|\eta_i| > \gamma] \leq 2 \int_{\gamma}^{\infty} \frac{\alpha}{2T} e^{-x\alpha/T} dx = e^{-\gamma\alpha/T} \tag{5.1}$$

Hence, with probability at most δ the additive error is at least $\frac{T}{\alpha} \ln \frac{1}{\delta}$. The final result follows by normalizing the error by the $LIS(\sigma) = \sqrt{T}/\beta$. □

Space and Time Analysis. The memory and time complexity for this approach are the same as the non-private algorithm used to compute the real length of the LIS. Therefore, using the Patience Sorting algorithm for example, the space and update time required are $O(LIS(\sigma))$ and $O(\log LIS(\sigma))$ respectively.

5.2 Decomposition Framework

The baseline approach introduces an additive error that grows linearly with the length of the stream. Therefore, for small LIS this error could dramatically degenerate the utility of this solution. The reason for this large perturbation noise is due to the fact that each individual element in the stream could affect all the

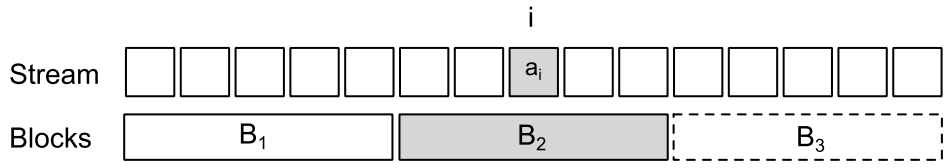


Figure 5.1: Block Decomposition example at time i: expired blocks (solid lines), active blocks (gray) and the future blocks (dashed lines).

possible outputs of the algorithm over the entire stream. This phenomenon could also occur for more sophisticated streaming algorithms that compute the LIS by using a small sketch of stream ([38, 63] for example). Although such solutions could reduce the space requirements, the use of a sketch does not directly reduce the error due to the perturbation noise since an element of the stream could still affect a large number of outputs (e.g. linear with the length of stream).

To overcome this problem, we decompose the computation of the LIS over segments of the stream. This intuition follows the idea proposed by Chan et al. [19] where a linear and binary decomposition frameworks are employed to privately compute the number of non-zero elements in a binary stream. Despite the similarity in these decompositions, the computation of the longest increasing subsequence is harder to achieve than the simple count function. For this reason, we study the utility loss in approximating the LIS inflicted by using the local information of the stream. Due to space limitation, we consider only an extension of the binary decomposition since it provides better utility with respect to the linear decomposition proposed in the original paper [19].

In our work, we investigate the implications of decomposing the LIS computation over blocks (i.e. stream segments) both from the utility and complexity perspective. It is important to note that the nature of the decomposition should be data-independent to avoid additional privacy cost. In principle, any algorithm $\mathcal{A}_{LIS}$ that computes the LIS (either exact or approximate way) can be used as a building block to compute the LIS on each stream segment so that the perturba-

tion noise required by the privacy mechanism can be reduced with respect to the direct use of $\mathcal{A}_{LIS}$. On the other hand, by limiting our computation on segments we introduce an approximation error.

In the rest of the section, we use the Patience Sorting algorithm [41] as a simple building block. We prove the reduction in the perturbation noise and the approximation error of our solution. We focus on this particular algorithm because it allows us to have an internal procedure that computes the exact length of the LIS over segments of the stream. In this way, we can directly measure how our decomposition impacts the exact solution. Since the original Patience Sorting algorithm computes not only the length of the LIS but also the elements forming the sequence, we use a modified version that only keeps the top element of the piles in the data structure as illustrated in the Algorithm1. In this way, we can compute the length of the LIS but using only $O(LIS(\sigma))$ space.

Before presenting our technique, we illustrate some concepts that will be useful in explaining our algorithm. A block $B = \sigma[j, j + b - 1]$ of size b represents a continuous segment of b symbols in the stream σ. Due to the dynamics of the data in the stream, a block assumes three different states over stream depending on the current time. At time i, the block B can be in one of the following states: **expired** hence the new arrival does not affect the block B (i.e. $j + b - 1 < i$), **active** when the new arrival is contained in the block B (i.e. $j \leq i \leq j + b - 1$) and **future** hence B contains only upcoming elements (i.e. $j > i$). An example of block decomposition of the stream is illustrated in Figure 5.1. The life cycle of a block B consists of starting as a future block, becoming active, and finally the block expiration.

5.2.1 Binary Decomposition

We start observing that in general the decomposition of the LIS over blocks may incur large approximation error. In fact, by simply dividing the stream into blocks

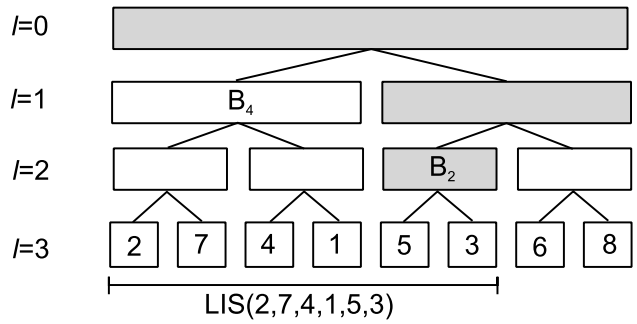


Figure 5.2: Binary Decomposition example. At time 5 (six symbols), the algorithm updates the active blocks (in gray). It computes the answer to the LIS query by summing the contributions of B_2 and B_4 containing the 2 and 4 most recent symbols respectively.

and combining the length of their LIS as a answer could lead to an approximation error that is proportional to the number of blocks used in the decomposition. To reduce this error, we develop a decomposition using variable length blocks, where the number of blocks in the stream decomposition is $O(\log T)$. We organize the blocks in a binary tree where at time i the tree has $\log i$ levels. Each level $l = 0, \ldots, \log i$ in the tree partitions the stream into disjoint blocks of length $i/2^l$. Figure 5.2 illustrates an example of binary decomposition of the stream.

Using this representation, each node k in the tree is associated with a block B_k and it stores the perturbed value of the $LIS(B_k)$. At any time i the algorithm updates the noisy LIS of the active blocks in the binary tree, and it answers the query $LIS(\sigma[0, i])$ as illustrated in Algorithm 7.

Algorithm Description. In the loop at lines 3-5, the algorithm updates the piles for the active blocks associated with the time i. In particular, the procedure `Update Piles` implements the Patience Sorting algorithm as in Algorithm 1, where in this case the update is performed independently on each active block B for any new coming element $\sigma(i)$. At lines 6-9, the noisy length of the LIS for each block that will expire is computed. At line 10, we compute the binary representation of $i + 1$ and let $i_1 < i_2 < \cdots < i_m$ be the positions of non-zeros in

Algorithm 7 Binary Decomposition

1: **procedure** BINARY DECOMPOSITION(T, α, σ)
Input: upper bound on the stream length T; privacy parameter α; event stream σ
Output: $\tilde{l}(\sigma)$ released longest increasing subsequence

2: **for** ($i = 0, 1, \ldots, T-1$) **do**
3: **for** (every active B at time i) **do**
4: UPDATE PILES($B, \sigma(i)$)
5: **end for**
6: **for** (every block B that will expire at time $i+1$) **do**
7: $LIS(B) \leftarrow$ number of piles for the block B
8: $\tilde{l}(B) \leftarrow LIS(B) + Lap(2 \log T/\alpha)$
9: **end for**
10: Let $i_1 < i_2 < \cdots < i_m$ be the positions of non-zeros in the binary representation of $i+1$
11: $\tilde{l}(\sigma) \leftarrow 0$
12: $k \leftarrow i$
13: **for** ($j = i_1, i_2, \ldots, i_m$) **do**
14: $B \leftarrow \sigma(k - 2^j + 1) \cdots \sigma(k)$ ▷ Retrieve the block to reconstruct the LIS
15: $k \leftarrow k - 2^j$
16: $\tilde{l}(\sigma) \leftarrow \tilde{l}(\sigma) + \tilde{l}(B)$ ▷ Sum the noisy contributions of the expired block B
17: **end for**
18: **Output** $\tilde{l}(\sigma)$
19: **end for**
20: **end procedure**

such representation. Then the answer for $LIS(\sigma[0, i])$ is computed by summing up the length of the LIS for the blocks containing the most recent $2^{i_1}, 2^{i_2}, \ldots, 2^{i_m}$ elements respectively. Therefore at each time i, the output result is obtained by adding the contributions of at most $\Theta(\log i)$ blocks in the loop at lines 13-17.

Privacy Analysis. We can observe that each element affects at most $\log T$ blocks; therefore, perturbing the LIS of each block with a random variable from $Lap(\log T/\alpha)$ is sufficient to satisfy the privacy requirement.

Theorem 5.2 (Binary Decomposition Privacy)**.** *The Binary Decomposition achieves α-differential privacy.*

Proof. In this decomposition, each element $\sigma(i)$ participates in the LIS of at most $\log T$ active blocks. Therefore, for any two neighboring streams the difference in L_1-norm of their outputs can be bounded by $\log T$. Therefore using Theorem **??**, it is sufficient to add to each LIS of each block a random variable from a Laplace distribution with parameter $\log T/\alpha$ to satisfy the privacy requirement. □

Utility Analysis. This decomposition with variable length blocks allows us to reduce the perturbation error due to the privacy mechanism. However, in this way we introduce an approximation error that depends on the number of blocks. We can observe that at most $O(\log T)$ blocks of variable length are needed to answer a LIS query. The utility results for this decomposition are reported below.

Lemma 5.3 (Binary Block Error Bound)**.** *Let σ be a stream of T symbols, and let $LIS(\sigma) = \frac{\sqrt{T}}{\beta}$, where β is positive. Without loss of generality we assume $T = 2^t - 1$, and we consider a partition of the stream σ into $B_0, B_1, \ldots, B_{t-1}$ non-overlapping blocks, where each block B_k is of size 2^k. Then in reporting the sum of the longest increasing subsequence in each block, $lis(\sigma) = \sum_{k=0}^{t-1} LIS(B_k)$, we incur the following approximation error.*

$$LIS(\sigma) \leq lis(\sigma) \leq \begin{cases} \log T \cdot LIS(\sigma) & \beta \geq 1 \\ (1 + \log \beta\sqrt{T}) \cdot LIS(\sigma) & \beta \in [1/\sqrt{T}, 1) \end{cases} \tag{5.2}$$

Proof. First, we start noticing the following lower-bound $lis(\sigma) \geq LIS(\sigma)$. In fact, the part of the real longest increasing subsequence which is contained in each block is at most the length of the longest increasing subsequence in the stream segment represented by the block. Second, we prove the two cases separately. For short value of $LIS(\sigma)$ ($\beta \geq 1$), we consider the case where each segment in each block is monotonic but none of them can be concatenated to form an increasing sequence in the entire stream. Then, we have that $LIS(\sigma) \geq LIS(B_k)$,

for $k = 0, \ldots, t-1$, which leads to $\log T \cdot LIS(\sigma) \geq \sum_{k=0}^{t-1} LIS(B_k) = lis(\sigma)$. For the case of long value of LIS ($\beta \in [1/\sqrt{T}, 1)$), we proceed as follows. Let j be a positive integer such that $2^{j-1} < \sqrt{T}/\beta \leq 2^j$. Therefore, for all the blocks B_k with $k \geq j$ we have that $LIS(B_k) \leq \sqrt{T}/\beta$, otherwise there exists a monotonic sequence which is longer than the longest increasing subsequence, hence we have a contradiction. Furthermore, due to the binary tree decomposition the sum of the length of the LIS for the blocks B_k with $k = 0, \ldots, j-1$ can be bounded as follows.

$$\sum_{k=0}^{j-1} LIS(B_k) \leq \sum_{k=0}^{j-1} 2^k = 2^j - 1 \leq 2\sqrt{T}/\beta \tag{5.3}$$

Therefore, the reported $lis(\sigma)$ can be upper bounded with the value below.

$$\begin{aligned} lis(\sigma) = \sum_{k=0}^{t-1} LIS(B_k) &\leq \sum_{k=0}^{j-1} LIS(B_k) + \sum_{k=j}^{t-1} LIS(B_k) \\ &\leq 2\sqrt{T}/\beta + (t-j)\sqrt{T}/\beta \\ &\approx \sqrt{T}/\beta(1 + \log \beta\sqrt{T}) \end{aligned} \tag{5.4}$$

This concludes the proof of the Lemma. □

Theorem 5.4 (Binary Decomposition Utility)**.** *The binary decomposition algorithm for computing the length of the longest increasing subsequence achieves the following utility results.*

$$\begin{cases} ((\log T - 1) + \frac{\beta \log^{3/2} T}{\alpha\sqrt{T}} \ln \frac{1}{\delta}, \delta)\textit{-useful} & \beta \geq 1 \\ (\log \beta\sqrt{T} + \frac{\beta \log^{3/2} T}{\alpha\sqrt{T}} \ln \frac{1}{\delta}, \delta)\textit{-useful} & \beta \in [1/\sqrt{T}, 1) \end{cases}$$

Proof. This decomposition has the advantage that the number of blocks combined in estimating the length of the LIS is only logarithmic which leads to an approximation error as shown in Lemma 5.3. This decomposition introduces a perturbation noise which is a sum of at most $O(\log T)$ i.i.d. Laplace random variables with parameter $O(\log T/\alpha)$. Let $\xi = \sum_k \eta_k$ denote the error due to the sum of

the Laplace random variables, we can use the result in Corollary 8.2 to bound this quantity. Choosing $\nu = \sqrt{\sum_k \frac{\log T}{\alpha}}\sqrt{2\ln\frac{2}{\delta}}$ with probability at least $1-\delta$, the quantity ξ is at most $O(\frac{\log^{3/2} T}{\alpha}\ln\frac{2}{\delta})$. Therefore, the final utility follows using the results from Lemma 5.3 and by normalizing this value by the $LIS(\sigma)$. □

Space Analysis. The space requirement for this approach is related to the number of active blocks that need to be updated and to the space complexity of the internal procedure. Due to the nature of the binary decomposition at any time i there are $\Theta(\log T)$ blocks that are active. Using a similar argument as in Theorem 5.4, we can show that the space complexity is $O(LIS(\sigma)\ln(\beta^2 LIS(\sigma)))$.

Theorem 5.5 (Binary Decomposition Space Comp.)**.** *Let $LIS(\sigma)$ be the length of the longest increasing subsequence in the stream σ, then the Binary decomposition framework has space complexity $O(LIS(\sigma)\ln(\beta^2 LIS(\sigma)))$.*

Proof. We begin by recalling that the internal procedure for computing the length of the LIS is the Patience Sort algorithm, where we keep only the top of the piles. At any time i in the stream, $\log T$ blocks are active, one in each level of the tree structure. Furthermore, let j be a positive integer such that $2^{j-1} < LIS(\sigma) \leq 2^j$. Therefore for the blocks in any level $i > j$ in the tree, we can upper bound their space requirements with $LIS(\sigma)(\log T - j + 1)$, since $LIS(\sigma)$ is the current length of the longest increasing subsequence. On the other hand, due to the nature of the binary tree the space required by the blocks below the level i is $2^j - 1$. Therefore the space complexity for this approach is $O(LIS(\sigma)(\log T - j + 1))$. Using the notion that $LIS(\sigma) = \sqrt{T}/\beta$ and $j = \log(LIS(\sigma))$, the previous requirements can be rewritten as $O(LIS(\sigma)\ln(\beta^2 LIS(\sigma)))$. □

Time Analysis. The total update time for this solution is related to the updates of the active blocks. Since at every time i there are $\Theta(\log T)$ active blocks, the update time is $O(\log T \log LIS(\sigma))$ using Patience Sorting algorithm.

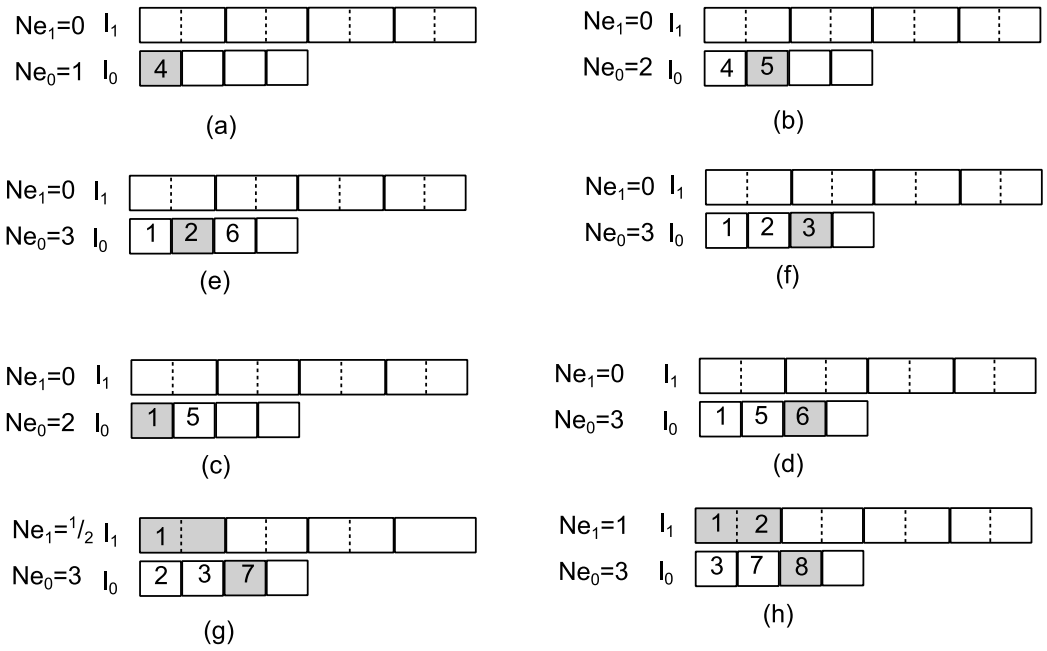


Figure 5.3: Running example of the Hierarchy mechanism on the input stream $4, 5, 1, 6, 2, 3, 7, 8$, $b = 4$ and $m = 2$.

5.3 Hierarchy Mechanism

In the previous section, we showed that the binary decomposition considerably reduces the perturbation noise in the final output compared to the baseline approach. However, such technique suffers from the fact that the computation of the LIS is generally hard to be decomposed in blocks leading in some cases to a large approximation error. To overcome this problem, we propose a new algorithm which computes the LIS over the stream by simulating the behavior of the Patience Sorting algorithm. In contrast to our previous approaches, this solution computes the length of the LIS by smoothing the impact of each element with the purpose of reducing the perturbation noise while achieving a good approximation ratio.

The main idea is to reduce the impact of those elements that stay too long in the LIS so that the total noise required by the privacy mechanism is decreased. Given

an integer $b > 0$, we construct a series of $m = \Theta(\ln \frac{T}{b})$ layers $l_0, l_1, \ldots, l_{m-1}$ with b buckets each, where at layer i each bucket contains 2^i elements. Given the series of elements with index $\{1, 2, \ldots, T\}$ in the stream, each layer simulates the behavior of the Patience Sorting algorithm where in this case the original piles are replaced with buckets that can contain multiple elements. In fact, at layer i the elements in the range $[(j-1)2^i + 1, j2^i]$ can be placed into the same bucket j. Intuitively, each layer has a different granularity, in fact l_0 keeps the exact top elements in the most recent b piles in the Patience Sorting algorithm, while l_1 keeps an approximation of the next $2b$ piles and so forth for the other layers. As the original algorithm, our procedure computes the length of the LIS by counting the number on non-empty buckets. In our case multiple elements may fall in the same bucket; therefore, we use a scaling factor equal to the length of the bucket to compute the contribution of each layer. Furthermore, in addition to insertion and replacement moves allowed in the Patience Sorting algorithm, we introduce an expiration move that forces elements that stay in a bucket at layer l_i for more than $2^i b$ iterations to be moved up to layer l_{i+1}. The algorithm computes the length of the LIS in the stream by adding the contribution at each layer. The code for this procedure is reported in Algorithm 8.

Algorithm Description. The algorithm starts initializing a set of m layers containing b buckets each, at lines 2-3. Within a layer i, each bucket is denoted with $P_i(j)$, for $j = 1, \ldots, b$ and it has size $2^i b$. In the main loop, lines 4-21, each new element coming in the stream is inserted in the first layer using the the same rule as the Patience Sorting algorithm, lines 5-7. In the inner loop at lines 9-13, the algorithm checks layer by layer to find the expired elements. When an expired element p in a pile $P_i(j)$ is found, the algorithm removes p and inserts it in the next layer. At line 14, the number of non-empty buckets for each layer is computed by normalizing the number of elements within each bucket with the corresponding bucket's size. In the loop at lines 16-19, the perturbation noise is applied to each count and finally the length of the LIS is returned.

Algorithm 8 Hierarchy Mechanism

procedure HIERARCHY MECHANISM(T, α, σ, b)
Input: upper bound on the stream length T; privacy parameter α; event stream σ; accuracy parameter b
Output: $\tilde{l}(\sigma)$ released longest increasing subsequence

$m = \Theta(\ln \frac{T}{b})$
Initialize each layer $l_i = [P_i(1), \ldots, P_i(b)]$ $i = 0, \ldots, m-1$ with b empty buckets
for $(i = 0, 1, \ldots, T-1)$ **do**
  Insert $\sigma(i)$ in l_1
  Find the largest $P_1(j)$ such that $P_1(j) \leq \sigma(i)$
  $P_1(j+1) = \sigma(i)$
  for $(i = 0, \ldots, m-1)$ **do**
    Let p be the element that expires at l_i
    Remove p from l_i and insert it in l_{i+1}
    Find the largest element in $P_{i+1}(j)$ such that $P_{i+1}(j) \leq p$
    $P_{i+1}(j+1) = p$
  end for
  Let Ne_i be the number of non-empty buckets at layer l_i
  $\tilde{l}(\sigma) \leftarrow 0$
  for $(i = 0, 1, \ldots, m-1)$ **do**
    $\widehat{Ne_i} \leftarrow Ne_i + Lap(mb/\alpha)$
    $\tilde{l}(\sigma) \leftarrow \tilde{l}(\sigma) + \widehat{Ne_i}$ ▷ Sum the noisy contribution of each layer
  end for
  Output $\tilde{l}(\sigma)$
end for
end procedure

We illustrate our hierarchy mechanism in the example below.

Example 5.6. *Consider the situation in Figure 5.3. When the first element arrives in the stream it is placed in the first bucket at* l_0 *as shown in (a). The second element that arrives is* 5*, since it is larger than* 4 *it is placed in the next bucket (b). The third element in the stream is* 1*. Since the insertion of the elements in*

the buckets follows the same rules as the Patience Sorting algorithm, we find the bucket that contains the smallest element larger than 1 *and insert this element in that bucket. Therefore, in our case,* 1 *overwrites* 4 *in the first bucket (c). At this point the length of the LIS is* 2*, as represented by the number of non empty buckets in* l_0*. The algorithm proceeds in a similar manner of the next three incoming elements (d),(e) and (f). After these new elements, the element* 1 *in* l_0 *is moved up to* l_1 *since it has been present in* l_0 *for more than* b *steps and the new incoming element* 7 *is inserted in* l_0 *(g). In the next step, the element* 2 *is moved up, and it is inserted in the same bucket with the element* 1*. At the same time the new element* 8 *is inserted in* l_0 *(e). The reported length of the LIS is obtained by summing the contribution of each layer. Layer* l_0 *contributes with* $Ne_0 = 3$ *and* l_1 *contributes with* $Ne_1 = 1$*. Hence the algorithm reports a length of the LIS of* 4 *while the exact length is* 5*.*

Privacy Analysis. In this algorithm the contribution of each element on the LIS is progressively decreased according to the layer in which the element appears. The privacy result for our hierarchy mechanism is reported in the following theorem.

Theorem 5.7 (Hierarchy Mechanism Privacy)**.** *The Hierarchy Mechanism achieves* α*-differential privacy.*

Proof. Given any two neighboring streams, we can observe that each element can affect at most m layers over the entire stream. In particular, at l_0 an element contributes to the LIS with a factor 1 for b times, at l_1 contributes with factor 1/2 for $2b$ and at the general level l_i contributes with factor $1/2^i$ for $2^i b$ times. Let $\mathbf{Ne}$ be the vector of contributions for each layer for the input stream $\sigma = a_1, \ldots, a_i \ldots, a_T$. Then, $\forall i \in [1, T]$ and $\sigma' = a_1, \ldots, a'_i \ldots, a_T$ we have that

$$\|\mathbf{Ne}(\sigma) - \mathbf{Ne}(\sigma')\| \leq mb \tag{5.5}$$

Then adding a random Laplace noise with parameter mb/α to the contribution of each layer i, is sufficient to satisfies α-differential privacy. Furthermore, us-

ing Corollary 8.2 we can see that the additive error introduced by noise is only $O(\frac{b}{\alpha}\log^{3/2}(\frac{T}{b})\log(\frac{2}{\delta}))$. □

Approximation Error. Our algorithm smooths the contribution of each element in the stream according to its layer leading to an underestimated value for the length of the LIS. The following Theorem summarizes the approximation ratio in the worst case.

Theorem 5.8 (Hierarchy Approximation Error)**.** *Let σ be a stream of length T, and b be the number of buckets in each layer of our algorithm. Then, the hierarchy mechanism returns a $(1-\frac{T-b}{T+b})$-approximation of the length of LIS.*

Proof. Let k be the length of the LIS over the entire stream. We begin by showing that this algorithm never overestimates the length of the LIS and then proceed by showing the error in the underestimate. To understand why this algorithm always reports a length of the LIS less or equal to the real length we consider the following case. Let us assume that there exists an element $\sigma(j)$ in a bucket at level $i > 0$ in our algorithm that differs from the Patience Sort. Since this element is extra in our algorithm it means that there is an element $\sigma(j')$, $j' > j$ that replaces $\sigma(j)$ in the exact Patience Sort. Since $\sigma(j') < \sigma(j)$, we have that in our structure $\sigma(j')$ has replaced another element $\sigma(j'')$. Due to the nature of our algorithm this operation could only occur in a layer $i' < i$, hence in replacing $\sigma(j'')$ with $\sigma(j')$ in our algorithm we have a larger loss of contribution than replacing $\sigma(j)$. Therefore we cannot have an overestimate length of the LIS.

Now, we examine the error in underestimating the length of the LIS. Consider a worst case scenario where only the first k symbols in σ contribute to the LIS, while the rest of the stream does not increase the length of the LIS. In this situation, as the stream proceeds the elements of the LIS that initially are in layer 0 are progressively moved up introducing a small additive error. Below, we quantify this error. Let $m = \log(\frac{T}{b}+1)-1$ be the number of layers in our structure,

then the maximum additive errors on the LIS is achieved when all the elements forming the LIS are in layer m. This quantity is computed as follows.

$$\sum_{i=1}^{m} \frac{k}{2^i} = k\left(\frac{T-b}{T+b}\right) \tag{5.6}$$

Hence the returned value from our algorithm is lower bounded by $LIS(\sigma)(1 - \frac{T-b}{T+b})$. This shows that our returned length $\tilde{l}(\sigma)$ satisfies the following inequality.

$$LIS(\sigma)\left(1 - \frac{T-b}{T+b}\right) \leq \tilde{l}(\sigma) \leq LIS(\sigma) \tag{5.7}$$

Therefore, our algorithm provides a $(1 - \frac{T-b}{T+b})$-approximation of the length of LIS. □

Space Analysis. Since this algorithm simulates the Patience Sorting algorithm by keeping only the top of the piles forming the LIS, it follows that the space complexity is linear with the length of the LIS in the stream $O(LIS(\sigma))$.

Time Analysis. For any new incoming element in the stream, the total running time is given by the cost required for updating each pile. There are at most $m - 1$ buckets, one for each level, that need update, where each operation requires $O(\log b)$ time. Since $m = \Theta(\log \frac{T}{b})$, the update time is $O(\log b \log \frac{T}{b})$.

This solution points out a strong connection between the approximation ratio and the noise required to achieve privacy. We can see that increasing b has a beneficial effect on the approximation ratio but on the other hand increases the privacy cost. In fact, as an extreme case using $b = T$ the algorithm returns the exact length of the LIS but incurs a large perturbation noise. Compared with our decomposition framework, this algorithm provides the user with a way to balance the approximation ratio and the noise due to the privacy mechanism.

Table 5.1: Summary of results for LIS query over entire stream.

Method	Error	Memory	Update Time
Baseline	$O(\frac{\beta\sqrt{T}}{\alpha}\ln\frac{1}{\delta})$	$O(LIS(\sigma))$	$O(\log\frac{\sqrt{T}}{\beta})$
Binary	$O((\log T-1)+\frac{\beta\log^{3/2}T}{\alpha\sqrt{T}}\ln\frac{1}{\delta})$ where $\beta\geq 1$ $O(\log\beta\sqrt{T}+\frac{\beta\log^{3/2}T}{\alpha\sqrt{T}}\ln\frac{1}{\delta})$ where $\beta\in[1/\sqrt{T},1)$	$O(LIS(\sigma)\ln(\beta^2 LIS(\sigma)))$	$O(\log T\log\frac{\sqrt{T}}{\beta})$
Hierarchy	$O((1-\frac{T-b}{T+b})+\frac{b\beta}{\sqrt{T}\alpha}\log^{3/2}(\frac{T}{b})\log(\frac{2}{\delta}))$	$O(LIS(\sigma))$	$O(\log b\log\frac{T}{b})$

5.4 Summary of Results

Table 5.1 summarizes the utility results of our proposed solutions. We can see that both our strategies outperform the baseline approach in many perspectives. We notice that the baseline approach incurs a large perturbation error which could dramatically compromise the utility. Specifically, the additive error in the baseline strategy grows linearly with the length of the stream. For the binary decomposition instead, we provide output-sensitive utility results showing the benefits of this technique for different lengths of LIS. Due to the use of disjoint blocks, this approach incurs a considerably smaller perturbation error with respect to the baseline solution. In fact, the dependency of the error with respect to the perturbation noise is only polylogarithmic in this case. Furthermore, we can observe that the decomposition framework has small space requirements and update time. In principle, the space and time complexity of this solution could be further improved by using more sophisticated algorithms (e.g. [38, 63]) as internal procedure instead of relying on the Patience Sorting. For count based statistic the binary decomposition has been shown very effective; however due to the nature of the LIS, this strategy incurs an approximation error. Our hierarchy approach specifically addresses the LIS problem by directly simulating the Patience Sorting algorithm. This procedure incurs a smaller computational time and it has small memory requirements. Comparing the worst case performance of this technique with the binary decomposition, we can observe that the decomposition framework is still superior leading to a smaller additive error with the same approximation ratio.

This result is due to the fact that the hierarchy strategy suffers when the LIS constitutes the initial part of the stream. In fact, as the execution proceeds the elements in the sketch are moved in higher level increasing the approximation error over the stream. However, we can notice that in real scenarios such situation is unlikely to occur because in many applications we can assume that the stream presents trends over time.

5.4.1 Extensions

In this section, we describe how to employ our developed techniques to solve real world problems.

Detecting trends in time-series data. Our proposed techniques can be extended to effectively detect trends in time-series data by restricting the computation of the LIS over windows in the stream. In fact, in monitoring applications, recent data is more important than distant data; therefore, using a sliding window W, we limit the computation of the LIS on the most W recent data. For example, a sudden increase of price in financial data will lead to an increment in the length of the LIS in the current window. Constraining the computation of the LIS on a sliding window of length W is beneficial both from the utility and complexity perspective. In fact, it has been shown in [7] that for the binary mechanism the use of a sliding window reduces the impact of the privacy to a factor that is independent from the length of the stream but it is only related to the size of the window W. A similar result can be also derived for the hierarchy mechanism, where in this case, the number of layers in the data structure depends only on the length of W rather than the entire stream.

Approximate String Matching. The problem of computing the LIS is a classical string matching problem that has been extensively studied in computational biology [40]. However, only few solutions have been proposed to privately match biological sequences. Generally, these approaches provide privacy and security

in matching strings by applying cryptographic techniques [60]. However, due to their high complexity these approaches may not be effective in real scenarios. In this setting, we believe that our solutions can be very promising by providing formal privacy guarantee and incurring a small computational overhead. Since the problem structure of the LIS is similar to other popular problems for computing string similarity measures (e.g. edit distance), we believe that our hierarchy approach could be a first step toward the design of efficient privacy preserving algorithms for matching strings.

5.5 Conclusions

In this chapter, we considered the problem of privately detecting trends in stream data. Specifically, we addressed the problem of computing the length of the LIS while protecting the presence of single event in the stream. We developed two different solutions that provide formal guarantee of privacy. The first approach approximates the length of the LIS by assembling local information computed on segments of the stream. The second approach constructs a small sketch of the stream by exploiting the structure of the problem. Using a rigorous analysis, we showed that these strategies provided significant benefits over the baseline approach.

For the future, we consider to investigate two possible research directions. First, we plan to further develop our extensions and turn our theoretical results into concrete algorithms to be applied to solve time-series monitoring and string matching problems. Second, our proposed solutions provide important insights about the privacy implications for computing complex ordered statistics. Therefore, we plan to better understand what kind of privacy sketching algorithms can benefit in this setting.

Chapter 6

Group Trip Planning Query

6.1 Problem Definition

In this section, we first formulate group route query problem and then prove the hardness of finding an optimal solution.

Definition 6.1 (Location Graph)**.** *Let $\mathcal{C} = \{c_1, c_2, \ldots, c_N\}$ be the universe of disjoint categories. A location graph $G = (V, E)$ consists of a set of nodes V and a set of edges E. Each node $v \in V$ represents a location, and an edge $e = (v_i, v_j)$ denotes a directed edge between v_i and v_j. Furthermore, each node v_i is associated with a set of categories $C(v_i) \in \mathcal{C}$ of the location represented by v_i, while each edge $e = (v_i, v_j)$ is associated with a positive number $\delta(e)$ representing the cost of traveling from location v_i to v_j.*

We represent G as a directed graph to better model the road-network constraints that are present in real scenarios, where the traveling cost between two locations may not always be symmetric. Our algorithmic solutions easily apply on undirected graphs as well. Furthermore, we use a set of categories for each node because in real settings a node might be a POI for multiple categories. For example, a shopping mall may have many restaurants, a post office, a theater etc. Our

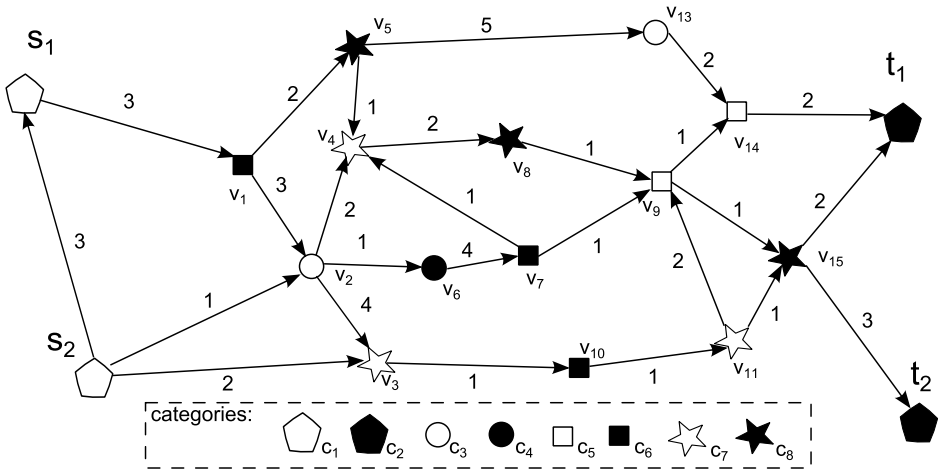


Figure 6.1: Location Graph, where each node is illustrated with a different shape representing the corresponding category.

solutions can also be adapted to consider hierarchical categories. An example of location graph with eight categories is illustrated in Figure 6.1. Only one category is associated with each node in this graph for visual simplicity.

Definition 6.2 (Route)**.** *A route $r = \langle v_0, v_1, \ldots, v_n \rangle$ is a path that starts at v_0 and ends at v_n and sequentially visits the nodes $v_1, v_2, \ldots, v_{n-1}$ through the edges in G.*

Definition 6.3 (Route Cost)**.** *Given a route $r = \langle v_0, v_1, \ldots, v_n \rangle$, the cost of r is defined as the sum of the cost of the edges visited in r as follows.*

$$\delta(r) = \sum_{i=1}^{n} \delta(v_{i-1}, v_i) \tag{6.1}$$

Definition 6.4 (Route Coverage)**.** *The route coverage for $r = \langle v_0, v_1, \ldots, v_n \rangle$ is defined as the set $\mathcal{C}(r) \subseteq \mathcal{C}$ of unique categories for the nodes visited in r, formally*

$$\mathcal{C}(r) = \bigcup_{i=0}^{n} \{C(v_i)\} \tag{6.2}$$

In our representation, graph G is populated by locations labeled with categories IDs. We assume that each user u_i starts from his/her location s_i and wants to reach a destination node t_i following a route without violating a cost constraint Δ_i. Each user u_i specifies a set of scores $\alpha_j^i \in [0, \alpha_{max}]$ for each category c_j representing how much he/she wants to visit c_j during the trip. We call this set of scores the *preference set* of u_i, which we denote with $\mathcal{P}(u_i) = \{\alpha_1^i, \alpha_2^i, \ldots, \alpha_N^i\}$. In addition, a weight w_i is associated to each user, and it determines the impact of user u_i in the final itinerary. This weight can also be used to model user's expertise regarding locations. For example, users living in the area are more reliable than tourists. On the other hand, we can use uniform weights if we wish to treat all group members equally. In the follows, we represent each user u_i as a five-tuple vector defined as $u_i = \{\mathcal{P}(u_i), \Delta_i, s_i, t_i, w_i\}$. In our problem, we consider the preference score for each user as a given parameter specified by the user. In principle, this score could be extracted automatically based on the user's history.

In this setting, we are interested in the problem of finding a feasible route for a set of users, so that they can follow and jointly visit a set of locations related to their individual preferences while maximizing the overall coverage of the preferred categories by the group. Below, we formalize these concepts by introducing the notion of *group route* for a set of users.

Definition 6.5 (Group Route)**.** *Given a set of users $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$, a group route (i.e. itinerary) $I = \langle v_0, v_1, \ldots, v_l \rangle$ for the users in $\mathcal{U}$ is a route in G, such that for each user u_i there exists a path $r_i = \langle s_i, \ldots, I, \ldots, t_i \rangle$ of length at most Δ_i that starts from s_i and ends in t_i. Furthermore, let $\mathcal{C}(I)$ be the set of distinct categories covered by the itinerary I, we define the weighted preference score $Ps(I)$ of the itinerary for the users in $\mathcal{U}$ as the weighted sum of the single node preference of each user, formally*

$$Ps(I) = \sum_{c_j \in \mathcal{C}(I)} \sum_{i=1}^{m} w_i \alpha_j^i \tag{6.3}$$

where $\alpha_j^i = 0$ if user u_i does not specify a preference for the category c_j.

The problem that we propose below is called *Optimal Group Route* query (OGR). Intuitively, we want to find a route that all users can share while maximizing their overall preferences among the specified categories. Consider for example the location graph in Figure 6.1, where two users u_1 and u_2 specify their starting points to be s_1, s_2 and ending points t_1 and t_2 respectively. Furthermore, let $\mathcal{P}(u_1) = \{\alpha_3^1, \alpha_5^1, \alpha_8^1\}$ and $\mathcal{P}(u_2) = \{\alpha_5^2, \alpha_6^2, \alpha_8^2\}$ be their preference sets, and $\Delta_1 = 15$, $\Delta_2 = 20$ be their respective mobility constraint. Then the itinerary $I = \langle v_2, v_4, v_8, v_9, v_{15} \rangle$ represents a possible route that both users can traverse during their paths from their start to their destination and covers part of the preferences of the users (c_3, c_5, c_8).

Problem 6.6 (Optimal Group Route Query)**.** *Given a set of users $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$, the Optimal Group Route Query $\mathcal{Q} = \langle \mathcal{U}, G \rangle$ determines a group route $I = \langle v_1, v_2, \ldots, v_n \rangle$ with maximum preference score $Ps(I)$ and such that all users can traverse. Specifically, for each user u_i, we return a path r_i of cost at most Δ_i starting from s_i and ending at t_i that traverses the computed itinerary I (i.e. $r_i = \langle s_i, \ldots, I, \ldots, t_i \rangle$).*

In deciding the group interest in visiting a single category, we use the user's weight to represent the influence of each user on the final decision. In this way, users with a larger weight have a stronger impact on the final score than users with a smaller weight. Furthermore, the cost constraints limit how much single users can detour from their routes to jointly visit the nodes that maximize the preference score for the specified categories.

Our proposed problem requires the location graph to be explored in order to search for the most profitable route that the users can traverse. Such formulation resembles the Hamiltonian Path problem. The following theorem states the connection between these two problems and show that the decision version of our OGR problem is NP-complete.

Theorem 6.7 (Hardness of OGR Problem)**.** *The problem of deciding if there exists a group route for a set of users that achieves preference score p on a location graph G is NP-complete.*

Proof. First, it is clear that OGR is in NP. Second, we show the hardness of the optimal group route problem by reducing the hamiltonian path problem (HP) on directed graph to it. Given an input graph $G = (V, E)$, a starting node s and an ending node t, the HP problem requires one to decide if there exists a hamiltonian path from s to t in G. Given an instance for HP, we create a new instance for OGR as follows. We consider a graph $G' = G$, where each node $v_i \in V$ is labeled with a unique category $C(v_i) = \{c_i\}$, and the cost for each edge is set to 1. Then, we consider the input query $\mathcal{Q} = \langle \mathcal{U}, G \rangle$ formed by only one user u with start and end points s and t respectively, with cost threshold $\Delta = |V| - 1$ and with preference set $\mathcal{P}(u) = \{\alpha_1, \alpha_2, \ldots, \alpha_{|V|}\}$, where $\alpha_i = 1$ for $i = 1, \ldots, |V|$. Therefore, it is clear that a group route I with preference score $p = |V|$ for user u exists if and only if the original graph G has a hamiltonian path from s to t. Since the reduction is polynomial with the size of the input graph G and given the fact that HP is NP-complete, it follows that OGR is NP-complete. □

We conclude this section by summarizing in Table 6.1 the frequent symbols used in the rest of the paper.

6.2 Algorithms

In this section, we illustrate our proposed solutions. We start by introducing some useful concepts that we heavily use in developing our solutions. Furthermore, we also present the preprocessing step used to accelerate our algorithms.

Table 6.1: Table of frequent symbols

Symbol	Description
$\delta(e)$	Cost of the edge e
$C(v_i)$	Set of categories for a node v_i
m	Number of users
α_j^i	Preference score of user u_i for category c_j
Δ_i	Mobility constraint for user u_i
w_i	Weight for user u_i
λ_j	Global preference score for category c_j
k	Number of non-zero preferences in the input query
$\pi(v_i, v_j)$	Shortest path from v_i to v_j
$\pi_p^*(v_i, v_j)$	Minimal path of profit p from v_i to v_j

6.2.1 Preprocessing Step

In the preprocessing step, we use the Floyd-Warshall algorithm [35] to compute for all the pairs of nodes (v_i, v_j) the shortest path $\pi(v_i, v_j)$ in the original graph from v_i to v_j. This information is used to construct the meeting graph (as we will explain below) in order to speedup the search of the optimal group route. We point out that this information is only related to the graph in input and it does not depend on the query issued. Therefore, given the location graph we can compute the shortest paths off-line and use this information when a query is issued. Although the preprocessing of all shortest pairs could be computationally intensive it is performed only once. Furthermore, such computation is standard in many optimal route problems, see [18] for example.

Once the input query $\mathcal{Q} = \langle \mathcal{U}, G \rangle$ is issued, we perform two preprocessing steps: computing the meeting graph and evaluating the global preference score for each category, to facilitate our algorithms.

Computing the Meeting Graph. In our problem formulation, we enforce the

constraint that all the users are able to visit the nodes forming the itinerary without violating their cost limit. Therefore in our solutions, we restrict the search of the optimal itinerary only on these special nodes. We denote these nodes as *meeting points*.

Definition 6.8 (Meeting Point)**.** *A node $v \in V$ is a meeting point for a set of users $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$, if for every user u_i there exists a path starting from s_i and ending in t_i of length at most Δ_i that passes through node v (reachable node) and covers some of the categories required by the users.*

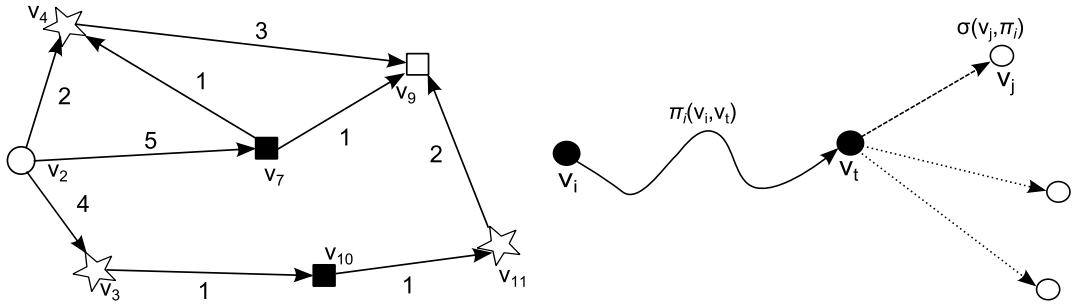
With this definition we construct from the original graph $G = (V, E)$ a subgraph $G_M = (V_M, E_M)$ as the induced graph from the meeting points of G where the edges in E_M are paths in the original graph that use only nodes that are reachable by all the users. We call this graph $G_M = (V_M, E_M)$ *meeting graph* for the users $\mathcal{U}$ in G. For example, in Figure 6.1, consider user u_1 starting from s_1 and ending in t_1 with $\Delta_1 = 15$, and u_2 starting from s_2 and ending in t_2 with $\Delta_2 = 20$. Then the reachable nodes for these two users are $\{v_2, v_3, v_4, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{15}\}$. Furthermore, let c_3, c_5, c_6, and c_7 be the categories required by the users. Then the meeting graph $G_M = (V_M, E_M)$ is defined as the subgraph induced by $V_M = \{v_2, v_3, v_4, v_7, v_9, v_{10}, v_{11}\}$ where v_6, v_8, and v_{15} are removed since they cover only the categories c_4 and c_8 that do not appear in the input query. In addition, the edges involving v_4, v_8, and v_{15} are replaced with new edges representing the shortest paths between their adjacent vertices that use only reachable nodes as shown in Figure 6.2(a). The algorithmic solutions presented later in this section have computational complexity related to the size of the meeting graph. Although in the worst case $G_M = G$, in real scenarios the meeting graph is considerably smaller than the original graph G. In our experiments section, we will illustrate this phenomenon and show how the size of G_M impacts the final performance. Furthermore, in computing the meeting graph we can test the existence of a solution. In fact, if G_M is empty, there is no possible

itineraries that the users can jointly visit. This result could be caused by mobility constraints too strong or/and by the specified categories in the query.

The construction of the meeting graph can be performed in an efficient way using the shortest paths information. We use a first phase, where for every node v, we check if every user is able to pass through v and reach his/her destination within his/her maximum cost constraint. For each user u_i we test if $\delta(\pi(s_i, v)) + \delta(\pi(v, t_i)) \leq \Delta_i$ holds. If such inequality is satisfied for all the users, then node v is a reachable node. Therefore, this step requires $O(|V|m)$, where m is the number of users in $\mathcal{U}$ and $|V|$ is the number of nodes in G. Furthermore, let $R = R_C \bigcup R_{NC}$ denote the set of reachable nodes which is given by the union of nodes covering some of categories in input query R_C and R_{NC} representing those that do not cover any specified query category. In the second phase, we remove the nodes in R_{NC} along with their edges and we add new edges in E_M if the removed node is on the shortest path between its adjacent nodes. This phase requires to process each node in R_{NC} and its incident edges at most once, hence it takes $O(|R_{NC}|^2)$ operations in the worst case (i.e. complete graph). Therefore the total running time in the worst case is $O(|V|m + |R_{NC}|^2)$. We point out that $|R_{NC}| < |R| << |V|$, and the entire process is very efficient in practice since real location graphs are quite sparse. In our experiments, we will measure the contribution of this step on the final running time.

Computing Global Preference Scores. An additional manipulation on the location graph is preformed using the information about the preference of each user. From Definition 6.5, given the set of users $\mathcal{U}$, for every category c_j we can compute a preference score λ_j (i.e. global profit) defined as the weighted sum of user preference as $\lambda_j = \sum_{i=1}^{m} w_i \alpha_j^i$. Intuitively, this measures the contribution of each category towards the global preference score.

Both the computation of the meeting graph and the scoring of the categories are performed on-line, when the query is issued.



(a) Example of meeting graph

(b) Extending minimal path $\pi^*(v_i, v_t)$ to node v_j

Figure 6.2: Example of meeting graph and minimal paths

Here, we also introduce a score for the node in the meeting graph. This concept will be used in our algorithms to evaluate the contribution in selecting a node to extend an itinerary. For each node v in the meeting graph, we introduce a score value $\sigma(v, I)$ that measures the increment of profit in an itinerary I if the node v is visited, which is the total contribution of categories covered by v but not covered by I already, formally:

$$\sigma(v, I) = \sum_{c_i \in C(v) - \mathcal{C}(I)} \lambda_i \tag{6.4}$$

This computation can be performed in $O(k)$ times, and in our solutions, we dynamically compute this score for only the nodes that are visited in the construction of the itinerary.

6.2.2 Dynamic Programming

In this section, we develop a *pseudo-polynomial* dynamic programming algorithm for computing the optimal itinerary. A possible way to tackle this problem consists in exploring all the possible paths that the users can traverse together while reaching their destination within their specific cost constraint and picking the one that maximizes the global preferences. This simple brute-force strategy incurs a

large computational cost since the number of possible paths to explore could grow exponentially with the number of nodes in the graph. However, we observe that it is not necessary to explore all the paths and we focus our search only on special shortest paths instead.

Consider the existence of an optimal itinerary I^* with optimal value $Ps(I^*)$ that starts from node v_i and ends in v_j in the meeting graph G_M. Then we can compute an equivalent itinerary of profit $Ps(I^*)$ by computing the shortest path between v_i and v_j satisfying the following properties: (1) the total profit score of the path is exactly $Ps(I^*)$ and (2) all the users are able to traverse such path starting from their source and ending in their destination. On the other hand, if such path does not exist then there is no itinerary of profit $Ps(I^*)$ between v_i and v_j in the meeting graph. Based on this observation, our algorithm aims to compute the shortest path for every pair of nodes with profit score exactly p. If such path exists and all the users are able to traverse it, then this path forms an itinerary of profit p for the users and it can be extended. Among all the resulting paths with different profit values p, we pick the one with maximum profit. In the rest of this section, we present the tools and concepts used in developing our solution.

Minimal Paths. To illustrate our approach, we start by fixing a source node v_i and use such node as a starting node for the group route in the meeting graph. Then, for every new node v_j encountered in traversing the meeting graph, we compute a set of paths called *minimal paths*, which are used to determine the candidate itineraries from v_i to v_j. The definition of minimal paths is presented below.

Definition 6.9 (Minimal Path)**.** *Given a profit score p, a path from v_i to v_j is called minimal path of profit p, denoted as $\pi_p^*(v_i, v_j)$, if it has minimal cost among all the paths from v_i to v_j with profit exactly p.*

$$\pi_p^*(v_i, v_j) = \arg_{\pi(v_i,v_j)|Ps(\pi(v_i,v_j))=p} \min\{\delta(\pi(v_i, v_j))\} \tag{6.5}$$

Given a profit value p, these paths have the property to be the shortest ones

achieving such profit, therefore it is sufficient to examine only these paths to determine the presence of an itinerary of such profit.

Example 6.10. *Consider the meeting graph illustrated in Figure 6.2(a), where the categories required in the query are c_3, c_5, c_6, and c_7. Furthermore, let v_2 be the candidate starting node for the itinerary and v_9 be the ending node. Given the input query, we can observe that the maximum profit achievable for the optimal route is $p = \lambda_3 + \lambda_5 + \lambda_6 + \lambda_7$. In this situation, there are two possible paths π_1 and π_2 that can achieve such profit. Where $\pi_1 = \langle v_2, v_7, v_4, v_9 \rangle$ and $\pi_2 = \langle v_2, v_3, v_{10}, v_{11}, v_9 \rangle$. Among them, π_2 is a minimal path since $\delta(\pi_2) < \delta(\pi_1)$. Therefore, if all the users can traverse π_2 without violating their distance constraint, we can extend π_2 and obtain the answer for the group route query. In our solution, we limit the search for the optimal route only to the minimal paths, avoiding the explicit enumeration of all the possible paths.*

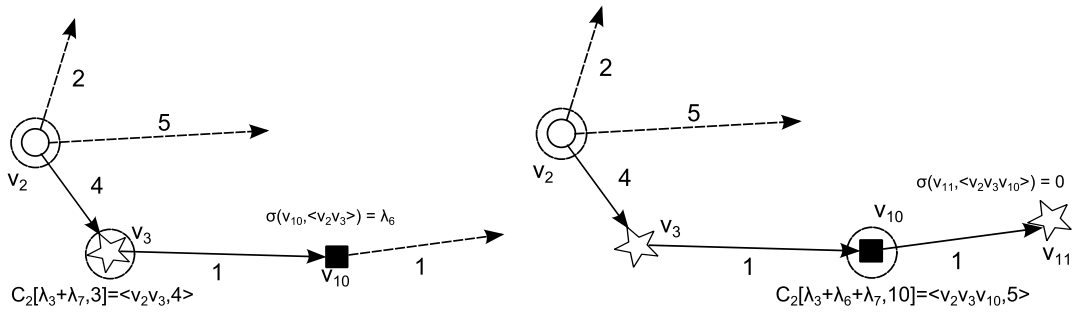
In general, for a given pair of nodes (v_i, v_j) and a profit value p, there exists multiple minimal paths between v_i and v_j achieving a profit p. We observe that give a node v_j and a profit value p, any two minimal paths π_1 and π_2 from v_i to v_j with profit p that cover the same set of categories are equivalent for us. Hence, we can just select one representative path among them. The following Lemma bounds the number of minimal paths stored by our algorithm for every new node encountered in our search.

Lemma 6.11 (Number of Minimal Paths)**.** *Given a starting node v_i, and the number of non-zero categories k specified in the input query $\mathcal{Q} = \langle \mathcal{U}, G \rangle$, the total number of minimal paths from v_i to any v_j is at most 2^k.*

Proof. Since the maximum number of categories that the users can select is k, for any node v_j the maximum number of minimal paths achieving profit p is equal to the number of subsets $S \subseteq \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ that sums to p. Since there are at most 2^k possible combinations of profit achievable, we use this as a upper bound on the number of minimal paths. □

Paths Construction. In the rest of the section, we develop a dynamic programming algorithm for computing the minimal paths. Let λ_i be the global profit score of a category c_i specified by the users in $\mathcal{U}$. Then, we can upper bound the maximum profit of the itinerary with $p_{max} = \sum_{i=1}^{k} \lambda_i$ since the global preference score of each category can be counted at most once. We use this information to guide our search for the optimal itinerary. In fact, for every $p \in [0, p_{max}]$, we compute all the minimal paths for any pair of nodes (v_i, v_j) in the meeting graph achieving a total preference score equal to p. In particular, for a starting node v_i, our algorithm fills a table C_i, where the entry $C_i[p, j] = \{P =< \{\pi_1, \ldots, \pi_l\} >, \delta\}$ represents the set P of the minimal paths from v_i to v_j with profit p covering different set of categories, and where δ represents the cost of such paths. Furthermore, since the preference score for the itineraries is defined on a unique set of categories, we need to keep track of the categories covered so far in our path extension. In fact, multiple nodes of the same category contribute only once to the profit for such category. To efficiently maintain this information, we index the paths in P using a hashmap. The computation of the minimal paths from node v_i is guided by the following observation. Given a set of minimal paths P ending at node v_t of profit value p (see Figure 6.2(b) for example), the set of paths for the neighbouring nodes v_j reachable with an edge from v_t can be computed as follows. For each minimal path π_i, we evaluate the profit score $\sigma(v_j, \pi_i)$ of each adjacent node v_j, then the path π_i can be extended to form a path of profit $p + \sigma(v_j, \pi_i)$ from v_i to v_j. Among all these resulting paths from v_i to v_j we update the entry $C_i[p + \sigma(v_j, \pi_i), j]$ by keeping only the shortest paths computed so far. An example is illustrated below.

Example 6.12. *Consider the graph reported in Figure 6.1, and let v_2 be the starting node for our procedure. Figure 6.3 illustrates two possible cases that occur during the minimal paths extension. In Figure 6.3(a), the minimal path from v_2 to v_3 is being extended to node v_{10}. Since v_{10} covers the category c_6 that is not currently covered by the minimal path to v_3, its score values with respect to the current minimal path is λ_6. Therefore, such node is selected to extend the current*



(a) Minimal Path to node v_{10} (b) Minimal Path to node v_{11}

Figure 6.3: Example of paths computation

minimal path to v_{10} with a total profit of $\lambda_3 + \lambda_6 + \lambda_7$. In Figure 6.3(b) when the node v_{11} is considered, its score value is 0 since it belongs to a category already covered by the current minimal path to v_{10}. Therefore, the profit for the minimal path reaching v_{11} is not increased.

Minimal Path Algorithm with Single Source. Our procedure is outlined in Algorithm 9. The idea of the algorithm is based on the Dijkstra's algorithm for one-source shortest path [27]. We keep a heap Q for all the possible entries in the table C_i, where the key of any entry $C_i[p, j]$ is the minimum cost path from v_i to v_j computed so far with profit p. We start by initializing the entry table and the first entry to insert in the heap at lines 2-7. In lines 10-30, we try to extend the current set of paths reaching v_t with profit p to the nodes v_j connected with an edge from v_t. In the loop in lines 12-17, we create an index CP which stores all the candidate path to v_j by profit value p_1. This profit is computed by evaluating the score of the node v_j with respect to the categories covered by the path π. In the next loop at lines 19-29, we try to update the minimal paths to the node v_j. In particular, we test if these paths have a lower cost than those already computed paths for node v_j (lines 22-27). Then, we can properly update the paths for the entry $C_i[p', j]$ and their respective keys in heap Q at line 28.

Lemma 6.13 (Correctness of Algorithm 9)**.** *When Algorithm 9 terminates, the entry $C_i[p, j]$ contains the minimal paths of profit p from v_i to v_j.*

Proof. In a similar way as in the shortest path algorithm, we can think the elements removed from the queue Q at line 10 forming a set $\mathcal{S}$ of pair (p, j) where the minimal paths of profit p to v_j are known. Therefore in the follows, we show that this statement holds during the entire execution of the algorithm. We proceed by proving it by induction on the size of $\mathcal{S}$.

- The base case consists in $|\mathcal{S}| = 1$, where $\mathcal{S} = (\sigma(v_i), i)$ corresponds to $C_i[\sigma(v_i), i]$ which is the minimal path from v_i to v_i with profit $\sigma(v_i)$. Therefore, the statement holds in this base case.

- Let (p, j) be the next entry inserted into $\mathcal{S}$ extended by an entry (p, t') added into $\mathcal{S}$ in the previous iteration. Furthermore, let $C_i[p, j]$ be the entry representing the path $\pi_p(i, j)$ extended from $C_i[p', t]$. Consider any other path $\pi_p(i, j)$ from v_i to v_j with profit p, and let (p'', z) be the first entry outside $\mathcal{S}$ such that $\pi_p(i, j)$ traverse v_z with score p''. Since in Algorithm 9 we remove the entries in Q according to their distance from v_i, we have that $\delta(\pi_p(i, j)) \leq \delta(\pi_{p''}(i, z)) \leq \delta(\pi_{p'}(i, t))$, therefore the path $\pi_p(i, j)$ is a minimal path.

□

Running Time for Single Source. Here, we analyze the running time for this algorithm with respect to the number of categories specified by the users and the size of the meeting graph $G_M = (V_M, E_M)$. First, we start analyzing the cost of testing if the current set of paths can be extended, in the loop 12-17 in Algorithm 9. Given each path π, we compute the score of the node v_j which requires inspecting the categories for the nodes used so far; therefore, $O(k)$ operations for each path are needed. In the loop at lines 19-29, we update the paths and we just

need to keep at most one path for each set of categories. Since, the minimal paths are indexed by their covered categories, the cost for performing such operation is $O(k)$. Furthermore, the set P of paths computed for each entry in the table grows exponentially only with the number of categories k. In fact, given a node v_j and profit value p, any two minimal paths π_1 and π_2 that lead to v_j with profit p and cover the same set of categories are equivalent for us during the entire algorithm. Hence, we can just store one representative for these paths. Therefore, using Lemma 6.11, we have that the overall cost for testing and updating the set of minimal paths is $O(k2^k)$. Second, the computational cost of our algorithm depends on the heap operations performed during the execution. Since the size of Q is fixed, we have at most $|V_M|p_{max}$ insert and delete operations. The most inner loop looks at each path in P, which are at most 2^k. Furthermore, for each node v_j associated to an entry we touch all its edges. Therefore, we have $|E_M|p_{max}$ set key operations in total. Hence, using a binary heap gives an overall running time of $O(p_{max}(|V_M| + |E_M|)\log(|V_M|p_{max}) + k2^k|E_M|p_{max})$. **From Minimal Paths to Itinerary**. Algorithm 9 computes a candidate itinerary from a fixed starting node v_i; however, in practice we do not know which node is the starting point for the itinerary in the meeting graph. Therefore, we use our procedure for every node v_i in the meeting graph. Although this strategy requires testing each node in the meeting graph, the size of such graph is considerably smaller than the original graph G, leading in many cases to small overall computational cost in practice. Running Algorithm 9, we fill a table C where entry $C_i[p, j]$ represents the minimal paths from v_i to v_j with profit p. Therefore, to retrieve the optimal itinerary I^* we proceed to examine the table C starting from the entry with maximum profit and for every pair of nodes (v_i, v_j), we pick the pair that can be extended to the destination of each user without violating the cost constraints. Once the itinerary $I^* = \langle v_i, \ldots, v_j \rangle$ is determined, we return a route r_i for every user u_i, where $r_i = \langle \pi(s_i, v_i), I^*, \pi(v_j, t_i) \rangle$ obtained by concatenating the shorted path from s_i to v_i with the returned itinerary I^* and the shorted path from v_j

to t_i. The complete procedure is illustrated in Algorithm 10. **Overall Running Time**. The cost for testing an itinerary has $O(m)$ computational complexity, since each user is involved in the test. Furthermore, in our approach there are at most $O(|V_M|^2 p_{max})$ itineraries on the entire meeting graph. Thus, the running time for this step is $O(m|V_M|^2 p_{max})$. Since the information about the pairs-shortest path is pre-computed in the preprocessing step, the overall complexity depends on the size of the meeting graph and the internal procedure in Algorithm 9. Iterating such procedure for every node in the meeting graph, we have total running time of $O(p_{max}|V_M|(|V_M| + |E_M|)\log(|V_M|p_{max}) + k2^k|E_M||V_M|p_{max})$. Although we can observe that the running time is exponential with the number of categories in the query, in practice we can think k as a small number with respect to the graph size and therefore such dependency is dominating the overall complexity. However, the running time is still exponential with the input size, since the dependency with respect to the number of bits to represent p_{max} is exponential. As a final note, our solution can be parallelized. In fact, we can distribute the computation of the itineraries in Algorithm 9 to multiple processes which could improve the performance.

6.2.3 Approximation Algorithm

The dynamic programming presented in the previous section takes advantage of the profit value p to search all the possible minimal paths in the meeting graph satisfying the mobility constraints for the users. Although in this way we avoid exploring all the possible paths, the algorithm could still incur high computational cost due to the large value of p_{max}. In fact the algorithm in Section 6.2.2 presents a pseudo-polynomial dependency in its running time with respect to the magnitude of the total maximum profit. In principle, users with heavy weight and large preference score could lead to large value of p_{max} degenerating the performance of the algorithm. In this section, we address the pseudo-polynomial part of the computa-

tional complexity of our dynamic programming, where the running time linearly increases with value of p_{max} rather than the number of bits used in its representation. For the approximation strategy, we use a similar technique as in FPTAS (Fully Polynomial Time Approximation Scheme) for the Knapsack problem [25]. Given ϵ an approximation parameter, our approach scales down the global preference score λ_i of each category c_i to $\hat{\lambda_i}$, so that the maximum score achievable p_{max} by any itinerary is a polynomial function of the number of categories k in the query input. The approximation algorithm is outlined in Algorithm 11, while the approximation ratio is proved in the following theorem.

Theorem 6.14 (ScaledDP approx)**.** *thmscale The ScaledDP algorithm returns a* $(1-\epsilon)$ *approximation of the optimal itinerary.*

Proof. Let $\hat{I}$ be the itinerary returned by the dynamic programming procedure on the scaled graph and I^* be the optimal itinerary in the original graph. Furthermore, let $Ps(I)$ be the preference score of the itinerary $\hat{I}$ obtained by scaling back its score, and let $\hat{p}_i$ denote the scaled score for the node v_i. Therefore, we have

$$Ps(I) = \left(\frac{\lambda_{max}}{\gamma}\right) Ps(\hat{I}) = \left(\frac{\lambda_{max}}{\gamma}\right) \sum_{v_i \in \hat{I}} \hat{\lambda}_i \tag{6.6}$$

Furthermore, since we obtain an exact solution using dynamic programming on the scaled graph we have that $\sum_{v_i \in \hat{I}} \hat{\lambda}_i \geq \sum_{v_i \in I^*} \hat{\lambda}_i$. Using this inequality and the fact that the floor operation decreases the scaled profit by at most one we proceed with the following series of inequalities.

$$\left(\frac{\lambda_{max}}{\gamma}\right) \sum_{v_i \in \hat{I}} \hat{\lambda}_i \geq \left(\frac{\lambda_{max}}{\gamma}\right) \sum_{v_i \in I^*} \left(\frac{\lambda_i}{\lambda_{max}}\gamma - 1\right) \tag{6.7}$$

$$\sum_{v_i \in I^*} \lambda_i - \left(\frac{\lambda_{max}}{\gamma}\right) \sum_{v_i \in I^*} 1 \geq Ps(I^*) - \lambda_{max}\frac{k}{\gamma} \tag{6.8}$$

Combining the last inequality with (6.6), we have that $Ps(I) \geq Ps(I^*) - \lambda_{max}\epsilon$, furthermore by the optimality of I^* we have that $Ps(I^*) \geq \lambda_{max}$. Hence we conclude that $Ps(I) \geq Ps(I^*)(1-\epsilon)$. □

Running Time. Since this algorithm uses Algorithm 10 as sub-procedure, we have that its running time is $O(\frac{k^2}{\epsilon}|V_M|(|V_M|+|E_M|)\log(|V_M|\frac{k^2}{\epsilon})+k2^k|E_M||V_M|\frac{k^2}{\epsilon}))$ which is polynomial in $1/\epsilon$.

6.2.4 Greedy Algorithm

The greedy algorithm that we propose constructs an itinerary by selecting the nodes in the meeting graph that can best increase the score of the itinerary. Intuitively, this approach aims to iteratively construct a feasible route by selecting the best node to visit in each iteration. This choice is performed in a greedy manner by computing the profit of each node with respect to the categories covered by the partial route constructed so far. Our approach is illustrated in Algorithm 12.

The procedure receives as input the original graph G, the set of users $\mathcal{U}$ and the meeting graph G_M pre-computed. Starting from an empty itinerary I, the algorithm computes all-pairs shortest paths within the meeting graph. In the loop in lines 5-20, the algorithm scores each node v in the meeting graph according to the categories in the node and those already covered by the current itinerary ($\mathcal{C}(I)$) as in Equation (6.4) and it selects the highest score σ. If such score is 0, all the nodes with some profit have been already visited on the itinerary I and then we can exit the loop and return (lines 8-10). Otherwise, the set of candidate nodes $V(\sigma)$ with profit σ are retrieved. Among these nodes the algorithm picks the closest node to the end of the itinerary constructed so far (line 13). Notice that, as the first node in the itinerary, among all nodes with maximum profit, we select the node with minimum average distance from all the starting points of the users. For the selected node, at line 14, the algorithm tests if it can be attached to the itinerary without violating the cost constraints of each user. At lines 15-18, the current itinerary is extended with the selected node, the cost and the categories covered by the itinerary are updated.

Running Time. In the greedy approach, we use a hash table to index all the nodes

in the meeting graph according to their score σ. The algorithm requires computing the all-shortest paths within the meeting graph and using the Floyd-Warshall algorithm it takes $O(|V_M|^3)$ time. Let k be the number of unique categories specified by all users in input (i.e. non-zero preferences), then the scoring process at line 6 takes $O(k|V_M|)$ time. Furthermore, in the loop in lines 5-18 each node is tested at most once for each user, therefore the total complexity for this algorithm is $O(km|V_M|^2 + |V_M|^3)$, where $|V_M|$ denotes the number of meeting points. In practice, since the size of the meeting graph is small compared to the original graph, the algorithm is very efficient. **Utility**. In this paragraph, we analyze the effectiveness of the greedy algorithm. Given an itinerary query $\mathcal{Q} = \langle \mathcal{U}, G \rangle$, recall k is the number of non-zero preferences specified by the users. The following theorem states the approximation ratio for our greedy solution.

Theorem 6.15 (Greedy Approximation)**.** *The solution returned by the Greedy approach is a* $(1/k)$*-approximation to the optimal solution.*

Proof. Let I_G be the itinerary computed by the greedy algorithm and I^* be optimal solution. It is clear that $Ps(I_G) \leq Ps(I^*)$, since our algorithm picks the point in a greedy way by their score. To show the underestimate error in the greedy algorithm, we proceed as follows. We consider the case where the greedy algorithm returns an itinerary I_G that contains only one node v that covers only one category with maximum score $\sigma = \max_{i=1,\ldots,k}\{\lambda_i\}$, while the optimal solution consists in an itinerary I^* that covers all the categories. Therefore, we have that

$$Ps(I^*) = \sum_{i=1}^{m} w_i \sum_{j=1}^{k} \alpha_j^i = \sum_{j=1}^{k} \lambda_j \leq k \cdot \sigma = k \cdot Ps(I_G) \tag{6.9}$$

Therefore, we have that $\frac{Ps(I^*)}{k} \leq Ps(I_G) \leq Ps(I^*)$. □

6.3 Extensions of our Solutions

In this section, we discuss some possible extensions which enable our solutions to handle more realistic group route queries. In particular, we consider the use of *POI specific quality measure*, *order constraints of the categories*, and *problem relaxation*.

POI Quality Measure. In real scenarios, users may prefer to visit POIs that are popular within the specified categories. For example, tourists in Rome will likely prefer to visit a popular and high-rated museum like the Borghese Gallery rather than an unpopular or isolated museum. Therefore to improve the quality of the planned route we can consider a POI's specific quality measure in our itinerary construction. A possible way to capture the popularity/quality notion associated with POIs is to use a finer granularity for the categories in our definition. For example, POIs in the same category could be further refined into sub-categories: popular and unpopular with respective scores. In this way, we could model the different contributions of the selected POIs on the final score for the places visited by the group.

Order Constraints for Categories. Our developed solutions are mainly focused on the optimality of the route, hence the order in which the categories are visited is computed such that the optimal preference score is achieved. Nevertheless, in real applications, users may want to specify a preferred order in which the categories should be visited. For example, a group of friends may prefer have dinner at some restaurants first and successively watch a movie in some theater. We can observe that the presence of order constraints on the categories reduces the search space for the optimal solution. In fact, among all the possible itineraries covering the input categories, only a small portion of them may satisfy the order constraints. In principle, we can adapt our solutions to take into account the specified constraints during the path construction process. The greedy approach can be easily adapted by testing the order satisfiability for the categories covered by the current itinerary

each time a node is attached to the partial solution. The extension of the dynamic programming solutions can be achieved by adapting the procedure in Algorithm 9 as follows.

We consider the path extension process for the `One Source Path` procedure, and we illustrate how such strategy can be easily modified to handle order constraints. For example, we can assume that users specify a total order constraint on the input categories $c_1 < c_2 < \cdots < c_k$, which requires that in the final itinerary the selected node in c_1 has to be visited before the node in c_2 and so on. Therefore, when a new node v_j is reached from the current set of paths P ending at node v_t, in the for-loop at lines 12-17 in Algorithm 9, we test if the presence of the category $c(v_j)$ on the path violates the specified order constraint. By examining the current set of paths P to v_t, we can easily select those that with the insertion of v_j satisfy the order constraint on the categories and use them to updated the entry $C_i[p + \lambda_{c(v_j)}, j]$. The rest of the algorithm will remain the same. In this way, we can enable users to specify order constraints on the categories they want to visit and solve their query by using our dynamic programming based solutions.

Problem Relaxation. We discuss two possible relaxations of our optimal itinerary query. We first show how to extend our proposed solutions to handle a multi-objective formulation of the optimal itinerary problem and then describe a post-processing phase to refine the itinerary at user level.

- In some cases, users are interested in combining both distance traveled and profit achieved as utility function transforming the group itinerary problem to a multi-objective optimization problem. In principle, we can adapt our dynamic programming based solutions to take into account this new problem formulation. For example, given a parameter $\theta \in [0, 1]$, we can assign to each itinerary I computed by our solutions a score as follows:

$$score(\theta, I) = \theta Ps(I) - (1 - \theta)\Delta(I) \tag{6.10}$$

where $Ps(I)$ is the profit of I and $\Delta(I)$ is the total distance traveled by the users using I. In fact, our dynamic programming strategies compute for each possible profit p an itinerary I with that profit and with minimum length. We can use the length of the itinerary for computing the total distance traveled by the users and finally evaluate the score in equation (6.10). In this way, by changing θ, users can decide the best itinerary according to its profit and length allowing them to maximize the overall preference and minimize the total travel distance.

- Our returned itinerary is shared among all the users, therefore it could occur that a user visits some locations that do not belong to his/her preference list. Therefore, to further enhance the quality of the returned itinerary, for each user we can consider a post-processing step that refines the itinerary according to each user's preferences. Specifically, for each user we construct a new itinerary by selecting the nodes, in the optimal itinerary returned by our solutions, that belong to the user's preference list. Then, we connect them by selecting the shortest path between them. This post-processing phase is very efficient, since the information about the all-shortest paths is pre-computed, the overall running time is $O(m|I|)$, where $|I|$ is the number of nodes in the itinerary. In this way, we guarantee that: (1) users share the itinerary on locations with common categories and (2) each user's traveled distance is minimized.

6.4 Experiments

In this section, we study the performance of our proposed algorithms. For simplicity, we denote by `DP`, `SDP`, and `Greedy`, the dynamic programming (Section 6.2.2), the approximation approach using scaled dynamic programming (Section 6.2.3), and the greedy solution (Section 6.2.4), respectively. We also tested

the brute-force algorithm mentioned in Section 6.2.2, however due to its extremely long running time, we do not show its results.

6.4.1 Settings

Location Graphs. In our experiments, we consider two datasets: `calmap`[1] [52], and `rome`[2]. The first dataset represents a real world graph, which has been obtained by combining the information about the California road-network and the collection of point of interests (POIs). The POIs are classified into 63 different categories, such as hospitals, airports, etc. In the original work of Li et al. [52], the location graph is undirected; however, in our experiments we manipulate the graph by adding two directed edges for each arc in the original graph. The final location graph has more than 100k nodes, each associated with one category, and 220k edges. The second dataset is generated from the real road-network of the city of Rome, with 3353 vertices and 8870 edges. Vertices correspond to intersections between roads and edges correspond to roads or road segments. For this dataset, we synthetically generate the POIs on the nodes by considering 60 categories according to a power-law distribution and we consider nodes covering multiple categories. We use this dataset to understand how different POI distributions and category density impact the performance of our algorithms. Our algorithms are implemented in Python and the results are obtained on a Intel Xeon machine at 2.3Ghz.

Algorithms setup. For our proposed approaches we use several settings to have a comprehensive analysis of their performance. If not specified in the text the default parameters for the algorithms assume the values reported in Table 6.2. In our simulations, we introduce some simplifications for facilitating the experiments. First of all, we defined a specific mobility constraint Δ_i for each user u_i. We

[1] `http://www.cs.fsu.edu/ lifeifei/SpatialDataset.htm`
[2] `http://www.dis.uniroma1.it/challenge9/download.shtml`

choose to define such value as a ratio of the length of the shortest path $\pi(s_i, t_i)$ between the user's starting point s_i and destination t_i in the location graph. In this way, we have a threshold value that depends on the specific user's route. Second, in generating user queries, we pre-compute the preference score λ for the categories specified by the users. In this way, each user's weight and preference sre directly expressed in the preference score. In our simulations, we randomly generate the preference score in the range $[40, 400]$. We select such range to ensure a down scaling effect for our approximation algorithm in all the settings. In fact, according to parameters in Table 6.2, we have to guarantee $\lambda_{max} \geq k/\epsilon$, since $k = 4$ and ϵ can be as small as 0.1 in some experiments. Evaluations with different user's weights are reported in Appendix **??**.

Metrics. In most cases, we report the absolute utility and running time of our algorithms, which is in log scale. In some evaluations, we report the *relative running time* and *relative utility* for the `SDP` approach with respect to the optimal algorithm `DP`. These two quantities are defined as follows. The relative running time is defined as the ratio between the running time for the approximate solution and the optimal algorithm. In a similar way, we define the relative utility as the ratio between the utility (i.e. preference score of the itinerary) achieved by the approximate solutions over the optimal value returned by `DP`. Using these metrics, we can clearly see the trade-off achieved by the approximation algorithm that we proposed. In many cases, we are also interested in the size of the meeting graph with respect to the original location graph. Therefore, we define the *relative size of the meeting graph* as $|V_M|/|V|$. Our evaluations are performed on random queries and the final results are obtained as the average of 50 runs. To obtain consistent results among all the settings, we discarded trivial queries, which result in meeting graphs that are either empty or contain nodes from only one category.

Organization. In our evaluation, we consider different settings to understand the impact of each parameter on our solutions. In particular, we distinguish three different types of parameters: *algorithm* dependent (approximation parameter ϵ),

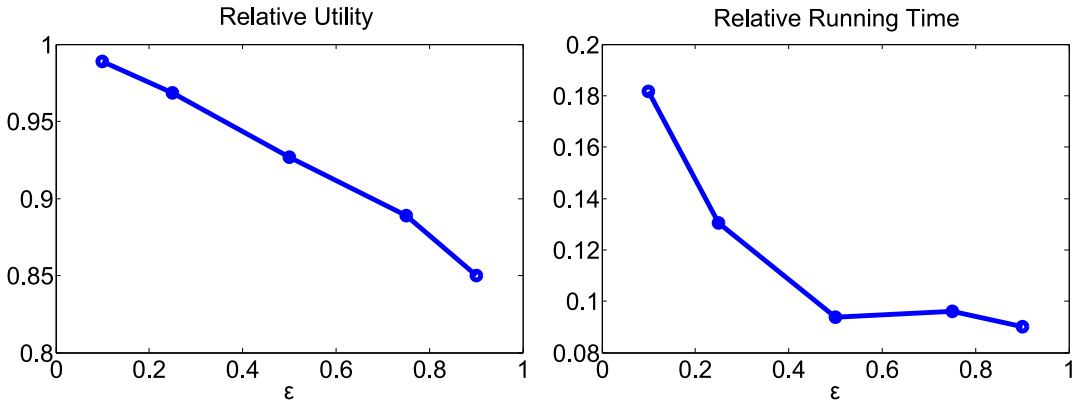
Table 6.2: Default Algorithmic Parameter Values

Parameter	Description	Value
m	Number of users	2
Δ_i	User mobility constraint	$1.5 \cdot \delta(\pi(s_i, t_i))$
k	Number of categories in the query	4
$[\lambda_{min}, \lambda_{max}]$	Global profit range	[40,400]
ϵ	Approximation parameter	0.5

workload dependent (number of users m, mobility constraint Δ_i, number of categories in the query k, scale of preference score λ, and category frequency in the query), and *data* dependent (graph size $|V|$, POI distribution and density). Due to the nature of these parameters, we divide our evaluation in two parts. First, we evaluate the impact of the algorithm and workload parameters, and scalability of our approaches on the real dataset `calmap`. Second, we use the synthetic dataset `rome` to evaluate our solutions with different POIs distributions and density which allows us to have full control on the distribution and density of the categories. Finally, we briefly illustrate the results obtained with some of the extensions proposed in Section 6.3.

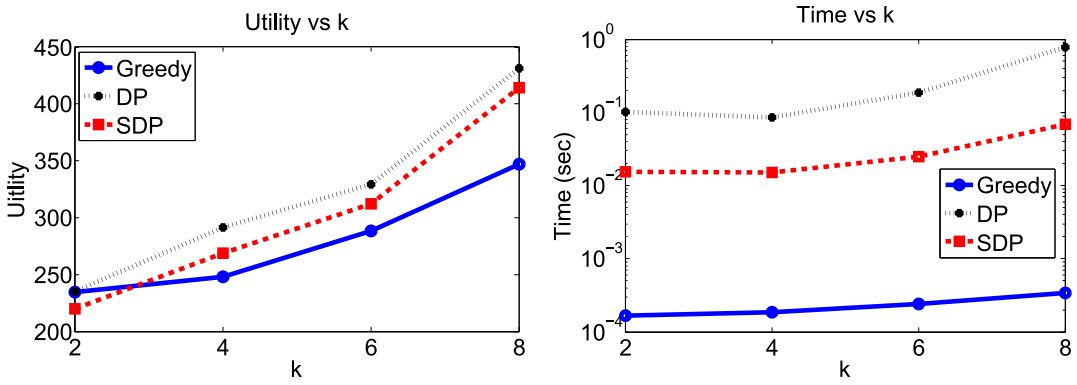
6.4.2 Results on Real Dataset

Impact of the ϵ parameter. We investigate the impact of the ϵ parameter on the performance of our approximation algorithm. In Figure 6.4, we present both the running time and the utility for the `SDP` algorithm as a fraction of the optimal `DP` algorithm. We observe that for increasing values of ϵ, the `SDP` provides a weaker approximation (Figure 6.4(a)), and requires smaller running time (Figure 6.4(b)). It is worth noting that the quality of the approximation obtained is considerably higher than what we could expect from the theoretical analysis. In fact, we have a considerable running time reduction and the utility results are always within the



(a) Relative Utility (b) Relative Running Time

Figure 6.4: Performance vs ϵ parameter

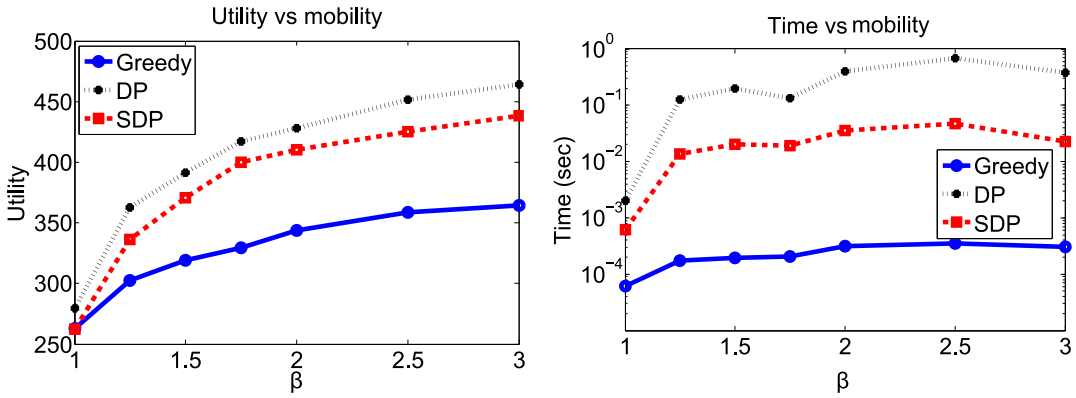


(a) Utility vs number of category (b) Time vs number of category

Figure 6.5: Performance vs k

85% of the optimal value. To obtain a good trade-off between running time and utility, we set $\epsilon = 0.5$ for the remaining experiments.

Impact of the parameter k. Figure 6.5 illustrates the impact of the number of categories k in the input query on the performance of our algorithms. From Figure 6.5(a), we observe that the utility of our approaches increases as k increases as we expected. We notice that the `SDP` algorithm provides high utility achieving

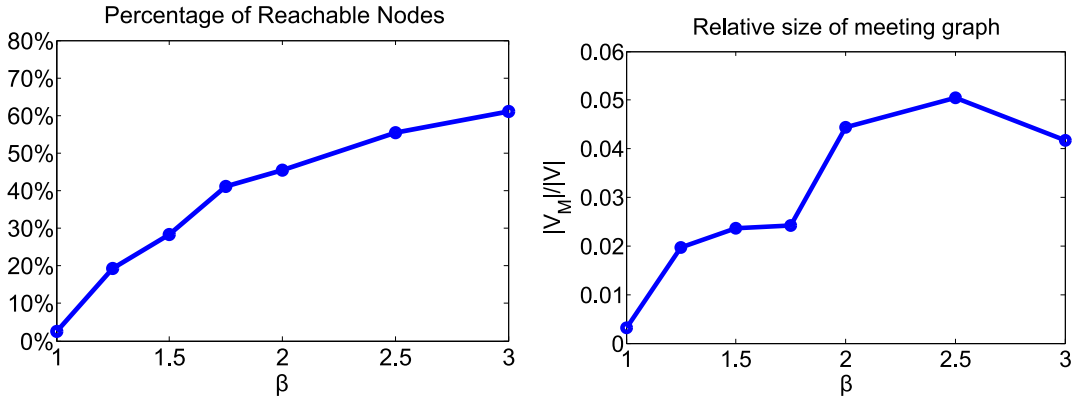


(a) Utility vs mobility (b) Time vs mobility

Figure 6.6: Performance vs mobility

profit score constantly above 90% of the optimal value. On the other hand, the `Greedy` approach shows its weakness. As the number of possible categories increases, the utility provided by `Greedy` drastically decreases reaching only less than 60% of the optimal value. This phenomenon is due to the greedy choice made by the algorithm. Intuitively, when the number of categories is large, the greedy algorithm could lose a large amount of profit since the decision of reaching first the nodes with larger profit could in some cases compromise the possibility of selecting nodes of the remaining categories. Figure 6.5(b) reports the running time for the algorithms when k is varying. As k increases the running time for `DP` and `SDP` increases, since a larger number of minimal paths could be constructed. The running time for `Greedy` slowly increases due to the low computational time required to evaluate the nodes score. In this setting, we can combine `SDP` and `Greedy` in a hybrid solution, where the greedy approach is employed for small k while `SDP` is used for large k. In this way, we can combine both beneficial aspects (small running time and good approximation ration) that come with these two solutions.

Impact of the mobility constrains. Figure 6.6 shows the impact of the mobil-



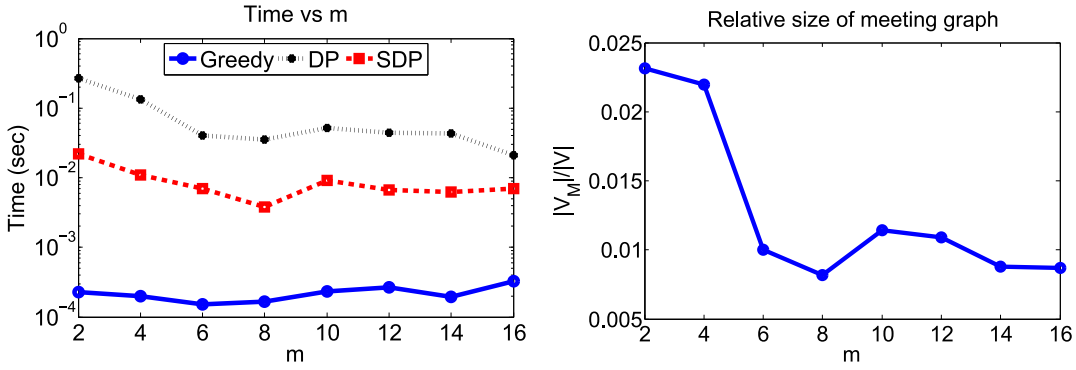
(a) Reachable nodes vs mobility

(b) Meeting graph vs mobility

Figure 6.7: Size of the meeting graph vs mobility

ity constraints on the performance of our algorithms. As we mentioned before, we use user specific mobility constraints defined as: $\Delta_i = \beta \cdot \delta(\pi(s_i, t_i))$, where $\delta(\pi(s_i, t_i))$ measure the length of the shortest path between the start node s_i and the destination t_i. Intuitively this reflects the detour or extra distance each user is willing to travel in order to accommodate the preference of his/her group members. In our experiments, we vary the parameter β allowing users to have larger mobility constraints. All the algorithms increase the profit of the itinerary computed as the user mobility increases (Figure 6.6(a)). Intuitively, less restrictive mobility constraints with large β values allow users to explore a larger portion of the graph, hence to cover more categories. This also increases the running time of our approaches (Figure 6.6(b)). In fact, from Figure 6.7, the number of nodes that can be reached by the users and the size of the meeting graph grow approximately linearly with the value of β. Despite the large number of nodes reachable by the users (Figure 6.7(a)), only a small portion of the nodes are covering the required categories leading to small meeting graph (Figure 6.7(b)).

Impact of the number of users m. In Figure 6.8, we report the performance of our solutions with different number of users in input. We can observe that



(a) Time vs users (b) Meeting graph vs users

Figure 6.8: Performance vs m

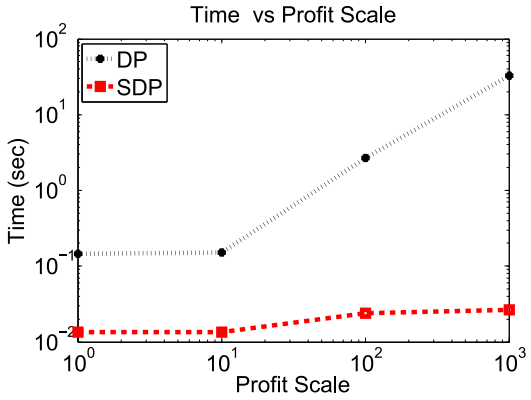


Figure 6.9: Running time vs preference scale

the running time for `DP` and `SDP` decreases as m increases (Figure 6.8(a)). This is due to the fact that a larger number of users in the query introduces stronger constraints on the meeting graph since they have to meet along the itinerary. In fact, we observe from Figure 6.8(b) that the size of the meeting graph decreases as more users are involved in the input query. The running time for the `Greedy` algorithm does not change considerably. In fact, the increment in the number of users is balanced with the reduction of the size of the meeting graph.

Impact of the preference score scale. Figure 6.9 illustrates the running time

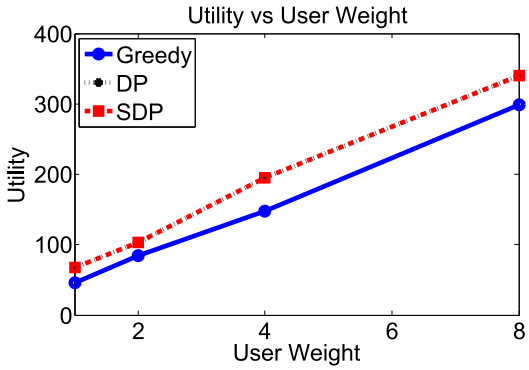
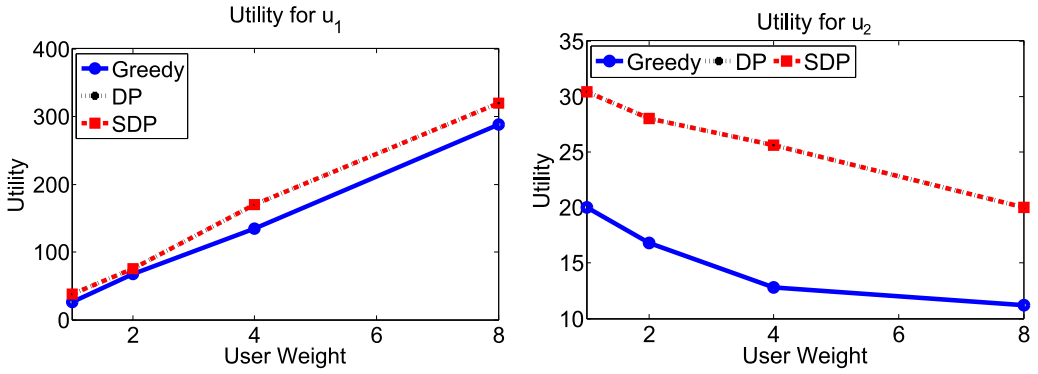


Figure 6.10: Utility vs User Weight

of DP and SDP with respect to different scale of preference score. As we mentioned before, the preference score λ incorporates the user's weight and preference. Therefore, in this experiment we observe how the combination of these two parameters affects the performance of our dynamic programming based solutions. As we shown in Section 6.2.2, the running time for DP grows linearly with the maximum preference score achievable with the given categories in the input query. We can also notice the beneficial effect of scaling the preference score in the SDP algorithm, where the running time for this approach does not present relevant changes. Hence, our SDP approach is robust against user's weight and preferences changes.

Impact of user's weight. In this setting, we explicitly consider the impact of the user's weight on the utility of our solutions. Specifically, we consider two users with weight w_1 and w_2 respectively and we study how changing the ratio between these weights affects the performance of our solutions. We introduce a parameter $\gamma = w_1/w_2$, that varies in the set $\{1, 2, 4, 8\}$, in this way we progressively increases the importance of user u_1 in the group decision. Figure 6.10 reports the overall profit of the itinerary computed for the group. As we expect, increasing the weight of u_1 increases the global utility as well. Looking at the utility for the individual users in Figure 6.11, we observe that for the first user the utility increases

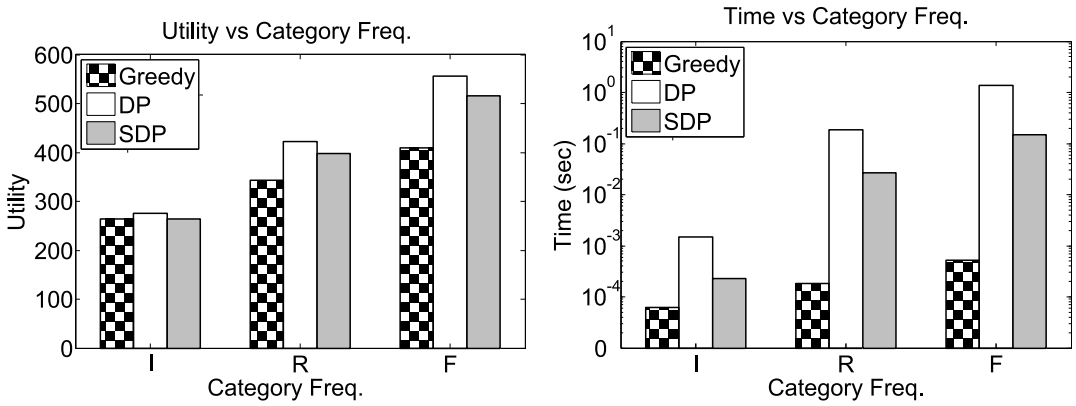


(a) User 1 utility (b) User 2 utility

Figure 6.11: Single user utility

(Figure 6.11(a)), while for the second user the usefulness of the itinerary tends to decrease (Figure 6.11(b)). This phenomenon is due to the fact that in choosing the group itinerary the first user's preferences have an high impact, hence some categories specified by the second user may be left out to cover more profitable POIs specified by the first user instead. In Section 6.3, we propose a possible extension of our solutions to enhance the utility for user u_2 that can be applied in this setting. From these results, we can notice that `DP` and `SDP` approaches have the same behavior and for both users. Compared to the `Greedy` solution, these approaches provide higher utility and are more robust since do not excessively penalize the utility for the second user.

Impact of the category frequency. In this experiment, we investigate the impact of the frequency of the categories on the performance of our algorithm. We construct three types of query workloads that differ in the frequency of the categories required to be visited: `I` infrequent, `R` random (mixed), and `F` frequent categories. Intuitively, for categories that occur frequently in the location graph it is more likely to find a set of candidate nodes that cover all the required categories. In fact from Figure 6.12(a), we can observe that the utility increases as the frequency of the categories in the query workload increases. The `Greedy` works well for



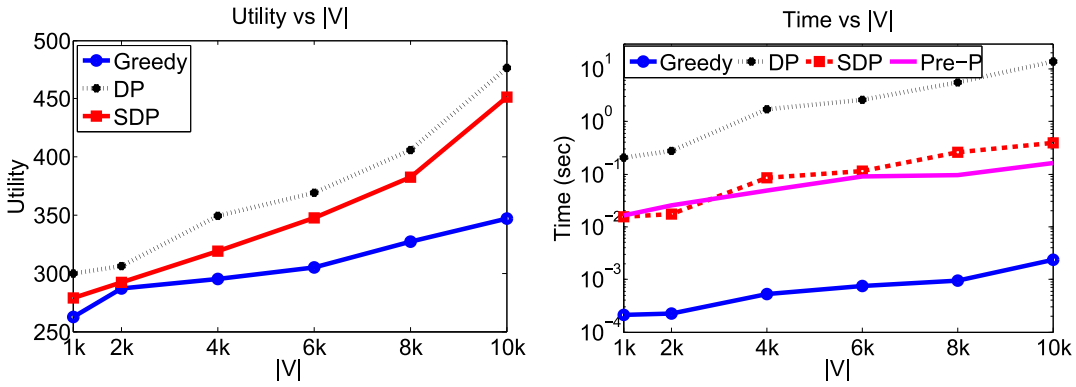
(a) Utility vs Query Freq.

(b) Time vs Query Freq.

Figure 6.12: Performance vs Category Freq

infrequent categories but when the frequency increases the gap with respect to the optimal solution increases. This is due to the increment of possible nodes to select in constructing the itinerary. On the other hand, the `SDP` provides good utility results across all the possible configuration of queries. Regarding the running time, as the frequency of the categories increases it is more likely to generate a larger meeting graph leading to higher running time, as shown in Figure 6.12(b). For all the proposed solutions increasing the frequency of the categories in the queries leads to higher running time.

Impact of the number of nodes $|V|$. We consider variable size of the location graphs in input. In particular, from the original graph we select connected subgraphs of size in the range from 1k to 10k nodes. The overall results are aligned with the theoretical analysis provided earlier in the paper. Although these dimensions may not be consistent with real world settings, these results give us a good understanding about the feasibility of applying our solutions to more practical scenarios. From Figure 6.13(a), we can observe that as the size of the graph increases all the approaches return itineraries of increasing profit. This is because more nodes can be selected in the construction process for the itinerary. In

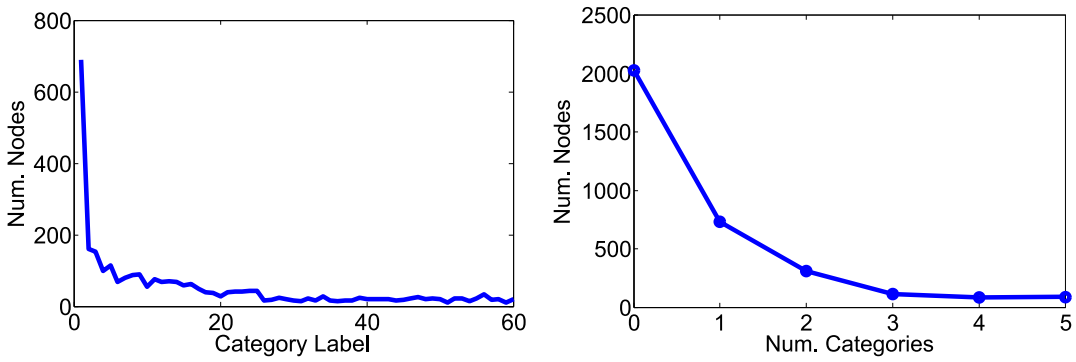


(a) Utility vs size

(b) Time vs size

Figure 6.13: Scalability

Figure 6.13(b), we illustrate the scalability of our solutions together with the pre-processing time from constructing the meeting graphs and scoring the categories (indicated with `Pre-P`). We observe that all our approaches are very efficient. The optimal algorithm is able to compute the itinerary for the largest graph in less that 15 seconds while the `SDP` takes less than 1 second. We observe that `Greedy` is more efficient compared to `DP` and `SDP`. However, due to the pre-processing running time the overall running time for the `Greedy` results to be almost the same as the time required by the `SDP` algorithm. Due to space limitation, we report the impact on the reachable nodes and meeting graph in Appendix **??**. We conclude this discuss by observing that our approaches scale well with the size of the graph in input and both `DP` and `SDP` can be parallelized. Furthermore, since the running time mostly depends on the size of the meeting graph; our solutions can benefit of user's limited mobility to efficiently compute the optimal itinerary on large graphs.



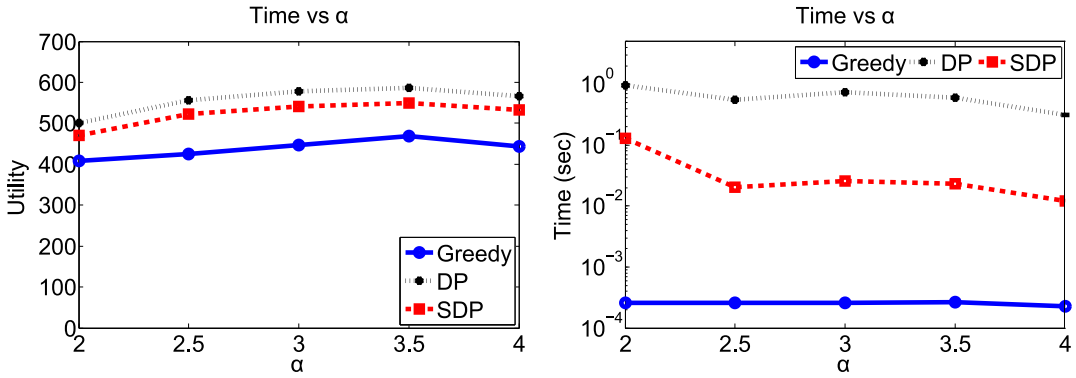
(a) POIs Category Frequency (b) POIs Categories per Nodes

Figure 6.14: Category frequency and density

6.4.3 Result on Synthetic Dataset

In our synthetic dataset generated from the road-network of the city of Rome, we tested the impact of the category distribution and density on the performance of our algorithms. We model the frequency of the POIs per categories as a power-law distribution of parameter α. This choice is motivated by the fact that in real scenarios we can observe a large difference between categories. For example, we can find many restaurants in a city but only few universities. Furthermore, on this location graph we allow nodes to cover multiple categories, for example a shopping mall may have many restaurants, a theater, and a post office. Hence a node can be labeled with multiple POIs category. In our experiments, we model the distribution of the number of categories per node using a Zipf's distribution of parameter α' where each node can cover up to 5 different categories. In Figure 6.14, we report an example of category frequency distribution and category density with parameter $\alpha = 2$ and $\alpha' = 1.5$ respectively. Specifically, from Figure 6.14(a) and Figure 6.14(b), we observe that there are about 700 nodes that are POIs in the first category and more than 300 nodes that cover two categories of POIs.

Impact of the category distribution. In our experiment we vary α to change



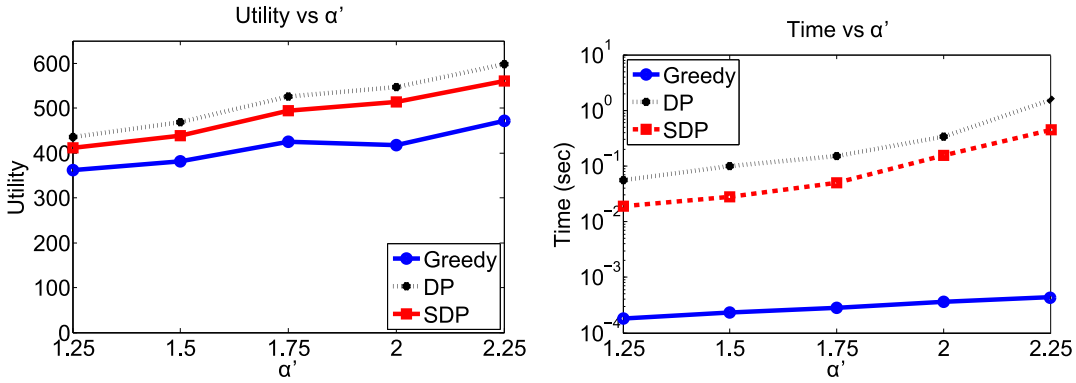
(a) Utility vs Category Distr.

(b) Time vs Category Distr.

Figure 6.15: Performance vs Category Distribution

the distribution of the categories and the results are reported in Figure 6.15. As α increases all the solutions do not present considerable changes in their utility (Figure 6.15(a)). Furthermore, for larger values of α the running time for the dynamic programming algorithms tends to decrease while the greedy solution does not show relevant changes (Figure 6.15(b)). Intuitively, with larger value of α the category distribution tends to become closer to uniform leading to a sparser meeting graph allowing both algorithms to compute the itinerary faster.

Impact of the category density. In this scenario, we vary the density of the categories on the nodes of the location graph. As the parameter α' increases more nodes are market as POIs and can be labeled with multiple categories. The performance of our solutions are reported in Figure 6.16. As the density increases, we can notice that the utility of our solutions increases (Figure 6.16(a)) since more nodes are marked as POIs hence more nodes can be selected to construct the itinerary. We can observe that `SDP` follows closely the optimal utility while the gap for the greedy approach increases. Similarly, the running time of our dynamic programming algorithms increase with the density (Figure 6.16(b)).



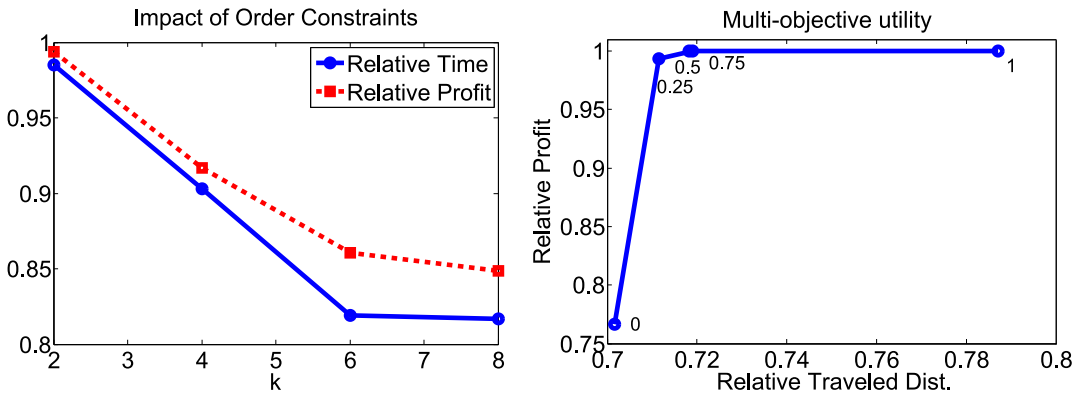
(a) Utility vs Category Dens. (b) Time vs Category Den.

Figure 6.16: Performance vs Category Density

6.4.4 Implementation of Extensions

We implemented two of the extensions for our `DP` approach proposed in Section 6.3: (1) we consider a constraint of total order for the k categories specified in input, and (2) we consider multi-objective utility that combines the profit and the distance traveled by the users. Below, we briefly illustrate the results obtained with these extensions on the `calmap` location graph.

Order constrains. We evaluate the impact of the order constraints on the running time and utility by comparing the results with those obtained on same queries without order constraints. Figure 6.17(a) reports both relative time and utility with different values of k. As we expected, the order constraints have a beneficial impact on the running time for our solution since the number of possible paths is limited to those that satisfy the specified order constraints. In particular, for $k = 8$ the running time for same query is reduced by almost 20%. On the other hand, requiring order constraints on the itinerary reduces the maximum utility that could be achieved since the "optimal" path may not satisfy the imposed order. In our experiments, we observe a utility reduction up to 15% on the utility of the optimal itinerary.



(a) Order Constraints (b) Multi-objective utility

Figure 6.17: Extension of our solutions

Multi-Objective utility. We consider a utility function as described in Equation (6.10), where the parameter θ sets the tradeoff between the profit of the itinerary and the average distance traveled by the users. In Figure 6.17(b), we report how the parameter θ impacts on the decision of the optimal itinerary. Both profit and distance are normalized between 0 and 1 to provide a clear representation of the trend. Setting $\theta = 0$ allows users to find an itinerary that minimize the distance traveled. In fact, we can observe that in this case the optimal itinerary has small profit but allows users to reach their destination within the 70% of their mobility. As θ increases, more weight is assigned to the itinerary's profit in the optimization. For $\theta = 1$, the optimal itinerary achieves the maximum profit and requires the users to travel up to the 80% of their mobility. From Figure 6.17(b), we can observe that for $\theta = 0.25$ the returned itinerary achieves a value of profit close to the maximum and incurs a small total distance traveled by the users making this itinerary a good candidate for the optimal solution.

6.5 Related Work

The search of optimal routes on graphs has its roots in the "Orienteering Problem" (OP). As in the sport game, a user starting from a point tries to visit as many nodes as possible within a given time constraint. Our proposed problem differs from the OP in many aspects. While OP is limited to a single user, in our formulation we are interested in computing a shared route for multiple users. Furthermore, in OP the profit in visiting a node is fixed, while in our problem such value depends on the users' preferences and on the sequence of nodes already covered, since the overall score of the route is computed on a set of unique categories. We refer the interested readers to [71] for a survey on the OP problem.

Li et al. [52] are the first to introduce the problem of finding route on spatial databases. In that work, the authors proposed a new problem called Trip Planning Query (TPQ), where each object in the database is associated with a location and a category label. Given a set of categories in input, a starting node and an ending node, the TPQ problem aims to find the shortest route from the starting to the ending node that passes through at least one point for each category. The authors in [52] showed that the TPQ problem is NP-hard, and they proposed a series of approximation solutions to tackle the problem. Compared to TPQ, our problem considers distance constraints and our goal is to maximize the profit for a group of users on the locations that are jointly visited.

Kanza et al. [49] proposed a new query problem, where the length of the route is bounded by a constraint and the goal is to maximize the profit in covering the categories of the nodes in the route between a given start and end point. The major differences between this approach and our work can be outlined as follows. First, in [49] an exact number of categories must be visited while in our case any subsets could represent a possible candidate. Second, in our case the score associated to each category depends on the user preferences while in Kanza's work the categories have the same score. Furthermore, we consider multiple users and

therefore the starting and ending points of the optimal shared route are not fixed as in [49] but depend on the locations where the users could meet. In successive works, Kanza et al. [47, 48] extended their approach by considering an interactive setting and introducing order constraints.

Sharifzadeh et al. [66] introduced a new extension of the TPQ problem, named Optimal Sequenced Route (OSR), where the goal is to find the shortest route from a given node that passes through an ordered sequence of locations. The authors proposed a series of pruning techniques to discard those locations that cannot be part of the optimal route. Chen et al. [20] proposed an extension of TPQ and OSR which considers the search of multi-rule partial sequenced routes (MRPSR). The authors showed that the MRPSR provides a unified framework that subsumes TPQ and OSR. In the original paper a series of heuristic algorithms have been developed to solve MRPSR queries. Recently, another extension of TPQ has been proposed by Li et al. [53]. In this formulation, the goal consists in computing the shortest route that covers a user defined set of categories with partial order constraints. The authors proposed two approaches namely backward and forward search to efficiently compute the optimal route.

Recently, Cao et al. [18] proposed a new route problem called Keyword-aware Optimal Route Search (KOR), which given a pair of nodes representing a start and end location in a graph, consists in finding the route that connects those two points such that a set of user-specified keywords is covered, a specified budget constraint is satisfied, and an objective score of the route is optimized. The authors in [18] proved the hardness of such formulation, and developed a series of approximation algorithms to solve the KOR problem. Our group trip query differs from both OSR and KOR in several aspects. We consider multiple users in the query, and we find the route of bounded length that maximizes the profit for the categories covered. Therefore, the solutions for OSR and KOR are not suitable in our setting.

B. Roy et al. [62] investigated the problem of computing an itinerary in an interactive way, where the user can provide feedback on the selected POIs to improve

the recommended itinerary.

When multiple users are involved in the route query, Hashem et al. [42] proposed a new problem called Group Trip Planning Query (GTP). Given a set of users with their start and end point in input, and a set of location categories, the GTP problem consists in finding a set of nodes belonging to the specified categories that minimizes the total traveled distance by the group. It has been shown by the authors in [42] that the GTP problem is related to the Group Nearest Neighbor query (GNN). Based on this observation, the authors proposed a series of heuristic approach to solve the GTP problem. Our approach is different from GTP since we consider a preference score associated to the categories rather than considering all the categories in the same way. Furthermore, we are interested in finding the best route in terms of preference score rather than distance.

6.6 Conclusion

In this chapter, we defined the problem of Optimal Group Route Query (OGR), which consists in finding the sequence of locations that the users can jointly visit and maximizes the preference of the overall group. We showed that this problem is NP-complete. We proposed an exact algorithm that computes such optimal route by avoiding the exploration of all the possible paths in the location graph. Our algorithm runs exponentially with k, where k denotes the number of categories specified by the users. We observed that in practice k assumes small values with respect to the size of the graph, which allows our algorithm to compute the optimal solution efficiently. Furthermore, we derived two approximation algorithms with bounded worst case approximation ratio that greatly reduce the computational cost. We showed how to extend our solutions to take in consideration quality measure for POIs such as popularity, order constraints for the specified categories, and several problem relaxations. Our experimental results showed the efficiency and effectiveness of our solutions both on real and synthetic data. Fu-

ture research directions consist in developing path construction algorithms to satisfy more flexible order constraints, considering dynamic updates on the map such as: adding/removing POIs, and travel distances (e.g. traffic scenario), addressing the challenge of synchronizing users in traveling together along the itinerary (e.g. consider waiting time), and introducing utility functions that consider the degree of coverage for the required categories. Furthermore, we aim to develop optimized data structures and code to handle large scale location graphs.

Algorithm 9 One Source Path

1: **procedure** ONE SOURCE PATH(G, G_M, v_i)
Input: graph $G = (V, E)$, meeting graph $G_M = (V_M, E_M)$, starting node v_i;
Output: Table of minimal paths

2: **for** (all v and p) **do**
3: $C_i[p, v] =< P = null, \delta = \infty >$
4: **end for**
5: $\sigma(v_i) \leftarrow \sum_{j \in C(v_i)} \lambda_j$
6: $C_i[\sigma(v_i), i] =< v_i, 0 >$
7: $Q.insert(key = 0, (v_i, C_i[\sigma(v_i), i]))$
8: **while** (Q is not empty) **do**
9: $(\delta, (v_t, < P, d >)) = Q.remove\_head()$
10: **for** (edges (v_t, v_j) in E_M) **do**
11: $CP \leftarrow < P = null >$
12: **for** ($\pi \in P$) **do**
13: $p_1 \leftarrow p + \sigma(v_j, \pi)$
14: $P_1 \leftarrow CP[p_1]$
15: $P_1 \leftarrow P_1 \bigcup < \pi, v_j >$
16: $CP[p_1] \leftarrow < P_1 >$
17: **end for**
18: $\delta_1 \leftarrow \delta + \delta(v_t, v_j)$
19: **for** ($p' \in CP$) **do**
20: $< P_2, \delta_2 > \leftarrow C_i[p', j]$
21: $P_1 \leftarrow CP[p']$
22: **if** ($\delta_1 < \delta_2$) **then**
23: $C_i[p', j] =< P_1, \delta_1 >$
24: **end if**
25: **if** ($\delta_1 == \delta_2$) **then**
26: $C_i[p', j] =< P_1 \bigcup P_2, \delta_1 >$,
27: **end if**
28: $Q.set\_key(key = \delta_1, C_i[p', j])$
29: **end for**
30: **end for**
31: **end while**
32: **return** C_i
33: **end procedure**

Algorithm 10 Dynamic Programming

procedure DP($G, G_M, \mathcal{U}$)
Input: graph $G = (V, E)$, meeting graph $G_M = (V_M, E_M)$, users $\mathcal{U}$;
Output: $\{r_1, r_2, \ldots, r_m\}$ user paths

  for (all v_i in V_M) **do**
    $C_i \leftarrow$ ONE SOURCE PATH(G, G_M, v_i)
  end for
  $p \leftarrow p_{max}$
  while $(p > 0)$ **do**
    for (all v_i, v_j in V_M) **do**
      $I^* = \langle v_i, v_{i+1}, \ldots, v_j \rangle \leftarrow C_i[p, j]$
      if (for all users $\delta(\pi(s_i, v_i)) + \delta(I^*) + \delta(\pi(v_j, t_i)) \leq \Delta_i$) **then**
        for all user u_i, $r_i = \langle \pi(s_i, v_i), I^*, (\pi(v_j, t_i) \rangle$
        return $\{r_1, r_2, \ldots, r_m\}$
      end if
    end for
    $p \leftarrow p - 1$
  end while
end procedure

Algorithm 11 Scaled Dynamic Programming

procedure SCALEDDP(G, G_M, ϵ)
Input: graph $G = (V, E)$, meeting graph $G_M = (V_M, E_M)$, users $\mathcal{U}$, approx. parameter ϵ;
Output: Itinerary $\hat{I}$, score $Ps(I)$

  $\gamma \leftarrow \frac{k}{\epsilon}$
  $\lambda_{max} \leftarrow \max_{i=1,\ldots,k} \lambda_i$
  scale the profit of each node $\hat{p}_i = \lfloor \frac{\lambda_i}{\lambda_{max}} \gamma \rfloor$
  $\hat{I} \leftarrow$ DP(G, G_M, ϵ)
  $Ps(I) \leftarrow \left(\frac{\lambda_{max}}{\gamma} \right) Ps(\hat{I})$
  return $\hat{I}$ and $Ps(I)$
end procedure

Algorithm 12 Greedy Algorithm

procedure GREEDY SELECTION($G, G_M, \mathcal{U}$)
Input: graph $G = (V, E)$, meeting graph $G_M = (V_M, E_M)$, users $\mathcal{U}$;
Output: Itinerary I

start with an empty itinerary I
$\mathcal{C}(I) \leftarrow null$
$\pi_{\mathcal{M}}(\cdot, \cdot) \leftarrow$ all-shortest paths within G_M
while (True) **do**
    Score each node v according to $\mathcal{C}(I)$
    $\sigma \leftarrow \max_v\{\sigma(v, I)\}$
    if ($\lambda == 0$) **then**
        Break
    end if
    let $V(\sigma)$ be the set of nodes in V_M with score σ
    let v_f and v_l be the first and last node in I respectively
    $v \leftarrow$ closest node to v_l in $V(\sigma)$
    if ($\forall$ users $\delta(\pi(s_i, v_f)) + \delta(\pi_{\mathcal{M}}(v_l, v)) + \delta(\pi(v, t_i)) \leq \Delta_i - \delta(I)$) **then**
        attach v to itinerary I
        $\delta(I) \leftarrow \delta(I) + \delta(\pi_{\mathcal{M}}(v_l, v))$
        $v_l = v$
        $\mathcal{C}(I) \leftarrow \mathcal{C}(I) \bigcup C(v)$
    end if
end while
return I
end procedure

Chapter 7

Conclusion and Future Work

In this dissertation, we addressed both privacy and social challenges that arise in modern database systems. In our work in privacy, we developed a series of algorithm techniques based on the formal notion of differential privacy to mine frequent sequential patterns and compute online statisitcs. Our solutions have beneficial impact of real world applications enabling knowledge discovery, data sharing, and data re-use. Regarding the social aspects, we proposed a new type of spatial query that allows users to find an optimal itinerary that can be jointly visited by the group. Our solutions provide a balance between single user's preference and constraints while maximizing the preference for the overall group of users. The outcome of this line of research can be beneficial for many popular and emerging location based services (e.g. Uber, Waze) by providing user's recommendation and supporting group query.

7.1 Summary

7.1.1 Privacy Contributions

Off-line Setting. In Chapter 3, we formalized the problem of mining sequential patterns from aggregated users data proving user-level privacy. In this setting, we studied the tradeoff between privacy requirements and utility of the mined patterns. We demonstrated that to achieve differential privacy requires to inject a perturbation noise proportional to the length of the sequence in input. Clearly, such perturbation may destroy the frequency of the patterns leading to poor final utility. To overcome this problem, we proposed a two-phase mining algorithm which first uses part of the privacy budget to extract useful information regarding the frequent patterns and successively transforms the original dataset in an compact representation to reduce the privacy impact on the utility. Our experimental evaluations showed the benefits of this approach on real datasets and also pointed out the ability of our technique in mining both frequent prefix and substring patterns. In Chapter 4, we proposed a new secure transformation technique for privacy preserving record linkage where the original string records are transformed into real vectors. We showed how our mining procedure is employed in the embedding process to construct a base that captures the structure of the original datasets required to be linked. Specifically, we introduced a new embedding technique that maps string records into real vectors and we studied how the distance between records is preserved in the new space. We also presented a matching procedure that allows approximate matching between records where the similarity function in the original space is measure using edit distance. Experimental evaluations demonstrate the benefits of our solution and the overall technique have been developed in our LinkIT tool [15].

On-line Setting. In Chapter 5, we proposed our privacy preserving solutions for computing ordered statistics over data stream. Specifically, we studied the prob-

lem of computing the length of the longest increasing subsequence while protecting the presence of single event in the stream. We showed how the privacy requirements in this scenario poses new challenges compared to traditional privacy solutions for streaming problems. In particular, our results provide an initial understanding about the relationship between memory and privacy requirements in this setting. Furthermore, we also pointed out some important applications that can greatly benefit from our solutions.

7.1.2 Social Contributions

In Chapter 6, we defined a new problem called Optimal Group Route Query (OGR), which consists in finding the sequence of locations in a location graph that a set of users in input can jointly visit such that the preference of the overall group is maximized. We demonstrated the hardness of this problem by showing that it is NP-complete. Nevertheless, we proposed an exact algorithm that computes such optimal route by avoiding the exploration of all the possible paths in the location graph. Our algorithm runs exponentially with k, where k denotes the number of categories specified by the users. We observed that in practice k assumes small values with respect to the size of the graph, which allows our algorithm to compute the optimal solution efficiently. Furthermore, we derived two approximation algorithms with bounded worst case approximation ratio that greatly reduce the computational cost. We showed how to extend our solutions to take in consideration quality measure for POIs such as popularity, order constraints for the specified categories, and several problem relaxations. Our experimental results showed the efficiency and effectiveness of our solutions both on real and synthetic data.

7.2 Future Work

In this dissertation, we addressed both privacy and social aspects in our modern society. We derived formal techniques and demonstrated the effectiveness of our approaches in several settings. Our results provide import insights in addressing fundamental problems in data privacy and social query. We believe that these possible future research directions could help in advancing modern database systems.

Extending Sequential Pattern Mining Techniques. The extraction of sequential patterns is crucial in many applications. However, in many real scenarios the sequential data produced in input comes with some noise. Consider for example the case of biomedical signals measured directly from the patient (e.g. hearth rate, blood pressure, etc.) such data often contains noise due to error measurement in reporting the signal. Another example comes from computational biology where punctual mutations may change some of the DNA basis in the sequence. Therefore this noise, either introduced by the measuring process or inherited by nature, may destroy the frequent sequential patterns in the original data. In this line of work, in [9] we initially proposed two definitions to capture the effects of the noise in the data. We pointed out possible scenarios where the mining of these patterns is central as well as the challenges in developing efficient mining algorithms. Future works include the extension of our privacy preserving prefix tree to mine noisy patterns, and developing privacy preserving techniques to handle genomic data.

Differential Privacy in the Streaming Setting. Our work on privacy preserving statistics over data streams [12] points out some of the important challenges in providing useful statistics while achieving privacy in a online setting. For the future, we consider to extend this work and propose two possible lines of research. First, we plan to further develop our solutions and turn our theoretical results into concrete algorithms to solve burst detection and string matching problems. These problems are extremely important in monitoring tasks(e.g. patient activity monitoring, finance, etc.) and biomedical informatics domain (e.g. genomic),

where the stream/string in input contains sensitive information that have to be protected. Second, our proposed solutions provide important insights about the privacy implications for computing complex ordered statistics. Specifically, we plan to better study the connection between privacy and space complexity as well as understand what kind of privacy sketching algorithms can benefit in this setting.

Extending Social Query. Modern mobile devices enable an easier user-to-user interaction creating opportunities for users to socialize. However, this phenomenon requires the need for modern database system to support queries that provide a service for a group of users. In this setting, a future research goal consists in understanding how users collaborate together to achieve a common goal. Specifically, we plan to extend our work on group query social queries for a variety of location-based services, such as: crowd-sourcing and ride sharing where the personal user's interest may be conflicting with the required tasks assigned to the group. We believe that the outcome of this research will enable a better user-to-user collaboration and ultimately provide tools to support new type of services.

Appendix

8.1 Statistical Tools for Multiple Laplace Random Variables

Lemma 8.1 (Sum Of Laplace Distributions [19]). *Let* $Y = \sum_{i=1}^{n} l_i$ *be the sum of* $l_1, \ldots, l_n$ *independent Laplace random variables with zero mean and parameter* b_i *for* $i = 1, \ldots, n$*, and* $b_{max} = \max\{b_i\}$*. Let* $\nu \geq \sqrt{\sum_{i=1}^{n} b_i^2}$*, and* $0 < \lambda < \frac{2\nu^2}{b_{max}}$*. Then* $Pr[Y > \lambda] \leq exp\{-\frac{\lambda^2}{8\nu^2}\}$

Corollary 8.2 (Measure Concentration [19]). *Let* Y*,* $\{b_i\}_i$*,* λ *and* b_{max} *defined as in Lemma 8.1. Suppose* $0 < \delta < 1$ *and* $\nu > \max\{\sqrt{\sum_i b_i^2}, b_{max}\sqrt{2\ln\frac{2}{\delta}}\}$*. Then* $Pr[|Y| > \nu\sqrt{8\ln\frac{2}{\delta}}] \leq \delta$

Bibliography

[1] Osman Abul, Francesco Bonchi, and Mirco Nanni. Never walk alone: Uncertainty for anonymity in moving objects databases. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 376–385, Washington, DC, USA, 2008.

[2] Miklós Ajtai, T. S. Jayram, Ravi Kumar, and D. Sivakumar. Approximate counting of inversions in a data stream. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 370–379, New York, NY, USA, 2002. ACM.

[3] Ali Al-Lawati, Dongwon Lee, and Patrick McDaniel. Blocking-aware private record linkage. In *Proceedings of the 2nd international workshop on Information quality in information systems*, IQIS '05, pages 59–68, New York, NY, USA, 2005.

[4] David Aldous and Persi Diaconis. Longest increasing subsequences: From patience sorting to the baik-deift-johansson theorem. *Bull. Amer. Math. Soc*, 36:413–432, 1999.

[5] Gennady Andrienko, Natalia Andrienko, Fosca Giannotti, Anna Monreale, and Dino Pedreschi. Movement data anonymity through generalization. In *Proceedings of the 2nd SIGSPATIAL ACM GIS 2009 International Workshop*

on Security and Privacy in GIS and LBS, SPRINGL '09, pages 27–31, New York, NY, USA, 2009.

[6] Raghav Bhaskar, Srivatsan Laxman, Adam Smith, and Abhradeep Thakurta. Discovering frequent patterns in sensitive data. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 503–512, New York, NY, USA, 2010. ACM.

[7] Jean Bolot, Nadia Fawaz, S. Muthukrishnan, Aleksandar Nikolov, and Nina Taft. Private decayed predicate sums on streams. In *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pages 284–295, New York, NY, USA, 2013. ACM.

[8] Francesco Bonchi, Laks V.S. Lakshmanan, and Hui (Wendy) Wang. Trajectory anonymity in *SIGKDD*, 13(1):30–42, August 2011.

[9] Luca Bonomi. Mining frequent patterns with differential privacy. *PVLDB*, 6(12):1422–1427, 2013.

[10] Luca Bonomi and Li Xiong. A two-phase algorithm for mining sequential patterns with differential privacy. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 269–278, 2013.

[11] Luca Bonomi and Li Xiong. Optimal group route query: Finding itinerary for group of users in spatial databases. *in submission*, 2015.

[12] Luca Bonomi and Li Xiong. Private computation of the longest increasing subsequence in data streams. In *Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT), Brussels, Belgium, March 27th, 2015.*, pages 270–277, 2015.

[13] Luca Bonomi and Li Xiong. Secure transformation via frequent grams for privacy preserving record linkage. *in submission*, 2015.

[14] Luca Bonomi, Li Xiong, Rui Chen, and Benjamin C. M. Fung. Frequent grams based embedding for privacy preserving record linkage. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 1597–1601, 2012.

[15] Luca Bonomi, Li Xiong, and James J. Lu. Linkit: privacy preserving record linkage and integration via transformations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 1029–1032, 2013.

[16] J. Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52:46–52, 1985. 10.1007/BF02776078.

[17] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 639–648, New York, NY, USA, 1996. ACM.

[18] Xin Cao, Lisi Chen, Gao Cong, and Xiaokui Xiao. Keyword-aware optimal route search. *Proc. VLDB Endow.*, 5(11):1136–1147, July 2012.

[19] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3):26:1–26:24, November 2011.

[20] Haiquan Chen, Wei-Shinn Ku, Min-Te Sun, and Roger Zimmermann. The multi-rule partial sequenced route query. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '08, pages 10:1–10:10, New York, NY, USA, 2008. ACM.

[21] 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 *Proc. of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, Beijing, China, August 2012.

[22] Rui Chen, Gergely Acs, and Claude Castelluccia. Differentially private sequential data publication via variable-length n-grams. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 638–649, New York, NY, USA, 2012.

[23] Rui Chen, Benjamin C.M. Fung, Noman Mohammed, Bipin C. Desai, and Ke Wang. Privacy-preserving trajectory data publishering by local suppression. *Information Sciences*, (0):–, 2011.

[24] Tim Churches and Peter Christen. Some methods for blindfolded record linkage. *BMC Medical Informatics and Decision Making*, 4(1):9, 2004.

[25] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[26] Graham Cormode, S. Muthukrishnan, and Süleyman Cenk Sahinalp. Permutation editing and matching via embeddings. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming,*, ICALP '01, pages 481–492, London, UK, UK, 2001. Springer-Verlag.

[27] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[28] Elizabeth Durham, Yuan Xue, Murat Kantarcioglu, and Bradley Malin. Quantifying the correctness, computational complexity, and security of privacy-preserving string comparators for record linkage. *Inf. Fusion*, 13(4):245–259, October 2012.

[29] Cynthia Dwork. Differential privacy. In *ICALP*, pages 1–12. Springer, 2006.

[30] Cynthia Dwork. Differential privacy: A survey of results. In *Theory and applications of models of computation*, pages 1–19. Springer, 2008.

[31] Cynthia Dwork. Differential privacy in new settings. In *SODA*, pages 174–183, 2010.

[32] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006*, pages 265–284, 2006.

[33] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 715–724, New York, NY, USA, 2010. ACM.

[34] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N. Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In *ICS*, pages 66–80, 2010.

[35] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345–, June 1962.

[36] Benjamin Fung, Ke Wang, Rui Chen, and Philip S Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (CSUR)*, 42(4):14, 2010.

[37] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Number v. 2. Cambridge University Press, 2009.

[38] Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *Proceedings of the eighteenth*

annual ACM-SIAM symposium on Discrete algorithms, SODA '07, pages 318–327, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[39] Anupam Gupta and Francis X. Zane. Counting inversions in lists. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '03, pages 253–254, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[40] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997.

[41] J. M. Hammersley. A few seedlings of research. In *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability*, pages 345–394, Berkeley, Calif., 1972. University of California Press.

[42] Tanzima Hashem, Tahrima Hashem, MohammedEunus Ali, and Lars Kulik. Group trip planning queries in spatial databases. In MarioA. Nascimento, Timos Sellis, Reynold Cheng, Jrg Sander, Yu Zheng, Hans-Peter Kriegel, Matthias Renz, and Christian Sengstock, editors, *Advances in Spatial and Temporal Databases*, volume 8098 of *Lecture Notes in Computer Science*, pages 259–276. Springer Berlin Heidelberg, 2013.

[43] G.R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(5), may 2003.

[44] Ali Inan, Murat Kantarcioglu, Elisa Bertino, and Monica Scannapieco. A hybrid approach to private record linkage. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 496–505, Washington, DC, USA, 2008.

[45] Ali Inan, Murat Kantarcioglu, Gabriel Ghinita, and Elisa Bertino. Private record matching using differential privacy. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 123–134, New York, NY, USA, 2010.

[46] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mapping into hilbert space. In *Conf. in modern analysis and probability*, volume 26 of *Contemporary Mathematics*, pages 189–206, 1984.

[47] Yaron Kanza, Roy Levin, Eliyahu Safra, and Yehoshua Sagiv. An interactive approach to route search. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '09, pages 408–411, New York, NY, USA, 2009. ACM.

[48] Yaron Kanza, Roy Levin, Eliyahu Safra, and Yehoshua Sagiv. Interactive route search in the presence of order constraints. *Proc. VLDB Endow.*, 3(1-2):117–128, September 2010.

[49] Yaron Kanza, Eliyahu Safra, Yehoshua Sagiv, and Yerach Doytsher. Heuristic algorithms for route-search queries over geographical data. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '08, pages 11:1–11:10, New York, NY, USA, 2008. ACM.

[50] VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[51] Chen Li, Bin Wang, and Xiaochun Yang. Vgram: improving performance of approximate queries on string collections using variable-length grams. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 303–314, 2007.

[52] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. On trip planning queries in spatial databases. In *Proceedings of the 9th International Conference on Advances in Spatial and Temporal Databases*, SSTD'05, pages 273–290, Berlin, Heidelberg, 2005. Springer-Verlag.

[53] Jing Li, Yin Yang, and Nikos Mamoulis. Optimal route queries with arbitrary order constraints. *IEEE Trans. on Knowl. and Data Eng.*, 25(5):1097–1110, May 2013.

[54] Ninghui Li, Wahbeh Qardaji, Dong Su, and Jianneng Cao. Privbasis: frequent itemset mining with differential privacy. *Proc. VLDB Endow.*, 5(11):1340–1351, July 2012.

[55] David Liben-Nowell, Erik Vee, and An Zhu. Finding longest increasing and common subsequences in streaming data. In *Proceedings of the 11th annual international conference on Computing and Combinatorics*, COCOON'05, pages 263–272, Berlin, Heidelberg, 2005. Springer-Verlag.

[56] Frank McSherry and Ilya Mironov. Differentially private recommender systems: building privacy into the net. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 627–636. ACM, 2009.

[57] Frank D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 19–30, New York, NY, USA, 2009.

[58] Asli Özal, Anand Ranganathan, and Nesime Tatbul. Real-time route planning with stream processing systems: A case study for the city of lucerne.

In *Proceedings of the 2Nd ACM SIGSPATIAL International Workshop on GeoStreaming*, IWGS '11, pages 21–28, New York, NY, USA, 2011. ACM.

[59] M.O. Rabin. *Fingerprinting by Random Polynomials*. Center for Research in Computing Technology: Center for Research in Computing Technology. 1981.

[60] S. Rane and Wei Sun. Privacy preserving string comparisons based on levenshtein distance. In *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, pages 1–6, Dec 2010.

[61] G Reinert, S Schbath, and MS Waterman. Probabilistic and statistical properties of finite words in finite sequences. *Lothaire: Applied Combinatorics on Words*, 2005.

[62] Senjuti Basu Roy, Gautam Das, Sihem Amer-Yahia, and Cong Yu. Interactive itinerary planning. In *ICDE*, pages 15–26, 2011.

[63] Michael Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *SODA*, pages 1698–1709, 2013.

[64] Monica Scannapieco, Ilya Figotin, Elisa Bertino, and Ahmed K. Elmagarmid. Privacy preserving schema and data matching. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 653–664, New York, NY, USA, 2007.

[65] Rainer Schnell, Tobias Bachteler, and Jorg Reiher. Privacy-preserving record linkage using bloom filters. *BMC Medical Informatics and Decision Making*, 9(1), 2009.

[66] Mehdi Sharifzadeh, Mohammad Kolahdouzan, and Cyrus Shahabi. The optimal sequenced route query. *The VLDB Journal*, 17(4):765–787, 2008.

[67] Han Su, Kai Zheng, Jiamin Huang, Hoyoung Jeung, Lei Chen, and Xiaofang Zhou. Crowdplanner: A crowd-based route recommendation system. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 1144–1155, March 2014.

[68] Manolis Terrovitis and Nikos Mamoulis. Privacy preservation in the publication of trajectories. In *Proceedings of the The Ninth International Conference on Mobile Data Management*, MDM '08, pages 65–72, Washington, DC, USA, 2008.

[69] Patrick Tucker. Mit tech review: Has big data made anonymity impossible?, 2010.

[70] Esko Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–211, January 1992.

[71] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1 – 10, 2011.

[72] Henan Wang, Guoliang Li, Huiqi Hu, Shuo Chen, Bingwen Shen, Hao Wu, Wen-Syan Li, and Kian-Lee Tan. R3: A real-time route recommendation system. *PVLDB*, 7(13):1549–1552, 2014.

[73] Xiaokui Xiao, Gabriel Bender, Michael Hay, and Johannes Gehrke. ireduct: differential privacy with reduced relative errors. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 229–240, New York, NY, USA, 2011.

[74] Mohamed Yakout, Mikhail J. Atallah, and Ahmed K. Elmagarmid. Efficient private record linkage. In *ICDE*, pages 1283–1286, 2009.

[75] Xiaochun Yang, Bin Wang, and Chen Li. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In *In SIGMOD Conference*, 2008.

[76] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982.

[77] Roman Yarovoy, Francesco Bonchi, Laks V. S. Lakshmanan, and Wendy Hui Wang. Anonymizing moving objects: how to hide a mob in a crowd? In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 72–83, New York, NY, USA, 2009.

[78] Hyoseok Yoon, Yu Zheng, Xing Xie, and Woontack Woo. Smart itinerary recommendation based on user-generated gps trajectories. In Zhiwen Yu, Ramiro Liscano, Guanling Chen, Daqing Zhang, and Xingshe Zhou, editors, *Ubiquitous Intelligence and Computing*, volume 6406 of *Lecture Notes in Computer Science*, pages 19–34. Springer Berlin Heidelberg, 2010.

[79] Chen Zeng, Jeffrey F. Naughton, and Jin-Yi Cai. On differentially private frequent itemset mining. *Proc. VLDB Endow.*, 6(1):25–36, November 2012.