User Customizable and Robust Geo-Indistinguishability for Location Privacy

Primal Pappachan
Pennsylvania State University, USA
primal@psu.edu

Chenxi Qiu
University of North Texas, USA
chenxi.qiu@unt.edu

Anna Squicciarini
Pennsylvania State University, USA
acs20@psu.edu

Vishnu Sharma Hunsur Manjunath
Pennsylvania State University, USA
vxh5104@psu.edu

ABSTRACT

Geo-Indistinguishability (Geo-Ind), based on Differential Privacy, is a popular privacy notion of privacy used for protecting individual's location data. Existing approaches, to generate a Geo-Ind satisfying obfuscation function, rely on a server, as this generation is computationally expensive. As a result, these obfuscation functions are not modifiable by users and any customization will lead to weakening of the Geo-Ind privacy guarantees i.e., violation of constraints in the function. A non-customizable obfuscation function can map an individual to an undesirable location, leading to poor quality of service. We present a framework called CORGI, i.e., CustOmizable Robust Geo-Indistinguishability, which allows users to customize an obfuscation function and ensure it is robust i.e., after user customization only minimal number of Geo-Ind constraints are violated. The experimental results on a real-world dataset demonstrate the effectiveness of CORGI in generating obfuscation functions that are more robust against customization by users, e.g., removing 14.28% of locations from the range of the obfuscation function leads to 18.58% and 3.07% Geo-Indistinguishability constraint violations, when the obfuscation function is generated by prior approaches and CORGI respectively.

1 INTRODUCTION

Many location obfuscation mechanisms have been proposed for protecting location privacy of individuals [17]. These approaches, often placed in the context of service provisioning, transform users' actual locations into obfuscated locations to protect their privacy while ensuring the quality of service. *Geo-Indistinguishability (Geo-Ind)* is one of the most popular privacy criteria used in location obfuscation mechanisms [2]. It extends the well-known *Differential Privacy (DP)* [10] paradigm to protect location privacy in a rigorous fashion. To satisfy Geo-Ind, if two locations are geographically close, their reported obfuscated locations will have similar probability distributions i.e., given an obfuscated location, it is hard for an adversary to distinguish a true location among nearby ones.

When Geo-Ind is used as the privacy criteria, the obfuscation function is formulated as a *Linear Programming (LP)* problem with a large number of Geo-Ind constraints. This complex LP is solved at a cloud server as users' devices have limited computation capability [19, 25, 28]. The obfuscation functions, generated using such a workflow, tend to be monolithic as it provides the same obfuscation range and the granularity of location sharing for all users. The obfuscation range is a set of locations from which an obfuscated location is chosen, and the granularity of the location determines the size/semantics of the location being shared (e.g., block, county). Users may have different privacy needs and utility requirements depending on the context and application scenario. Prior work [14, 16] has looked at customizing the obfuscation range to provide users' customizability based on their own privacy/utility needs. However, they focused on statistical releases of data and not point queries that are used for sharing location data. [7] extended this and applied it to location privacy, where they represented the possible locations of a user and their indistinguishability requirements using nodes and edges in a policy graph. Their goal is to ensure *Geo-Ind* for any two connected nodes in the graph, and to achieve this, they apply DP-based noise to latitude and longitude independently. However, their approach is best suited when locations can be neatly categorized, i.e., indistinguishability among multiple locations in the same category (e.g., restaurants). Also, it does not allow specific customization of an obfuscation function, i.e., remove my home and office from the obfuscation range.

There are several challenges to be addressed in developing such a framework that allows users to customize location obfuscation mechanisms generated by an untrusted server. The first challenge is specifying the customization parameters. Users should be able to denote their preferred granularity of location sharing and preferences for obfuscation range. Note that the user preferences contain private information and should not be shared with the server that generates the obfuscation function. By allowing users to remove some of the locations from the obfuscation range, users become active participants in their location-sharing tasks. However, users' edits to shareable locations add a significant risk of violating the Geo-Ind privacy guarantees of the original obfuscation function. The second challenge, therefore, pertains to considering the function's robustness, where robustness is the property by which the obfuscation function maximally satisfies Geo-Ind constraints after user customization. If the user customizes a non-robust function, then an adversary with knowledge of the prior probability distribution of the user check-ins in the area (from publicly available information) may be able to eliminate location(s) from the obfuscation range (as they are improbable) and hence increasing their chances of correctly distinguishing the user's real location from other nearby locations. The third challenge is performing these operations efficiently, as generating such a customizable obfuscation function is an expensive optimization problem with many constraints. Efficiency is also a challenge when the user updates their granularity of sharing, and a new obfuscation function has to be generated.

Series ISSN: 2367-2005

10.48786/edbt.2023.55

We propose a new framework called, *CORGI* (CustOmizable Robust Geo Indistinguishability), for generating location obfuscation with strong privacy guarantees that effectively allows users to balance the trade-off among privacy, utility, and customization. CORGI uses a tree structure which is a semantic representation of an area of interest; that assists users in specifying their customization preferences. The preferences selected by the user are used to select the obfuscation range and granularity of location sharing. CORGI utilizes an untrusted server for performing the computationally heavy task of generating the obfuscation function while ensuring the user's privacy. In order to protect the privacy of the user, the customization preferences are only selectively shared with the server e.g., only the number of locations to be removed from the obfuscation range and not the exact locations. The server in CORGI generates a *robust* obfuscation function, which would satisfy the *Geo-Ind* requirements after user customization. To generate this *robust* function efficiently, CORGI minimizes the number of constraints by using a *graph approximation* approach. We describe this workflow in more detail in Section 2.2. The experimental results on a real dataset show that the robust obfuscation function generated by CORGI is customizable with only a minimal number of constraint violations compared to the traditional approaches, which are not robust against customization.

The main contributions of this work are as follows:

- ▹ We propose *a tree-based approach* to assist users in specifying customization preferences. This improves the utility of location reporting as the number of locations in the obfuscation function is lower than traditional non-hierarchical approaches [21].
- ▹ We present a *customization preferences model* which is expressed in the form of Boolean Predicates and is selectively shared with the server for the purpose of generating an obfuscation function. Our customization model is more expressive than the prior work [7] which focuses on category-based privacy i.e., indistinguishability among multiple locations of the same category (e.g., restaurants).
- ▹ We develop a new framework for generating obfuscation functions that are *robust* against user customization which includes *a graph approximation* method to reduce the number of constraints and thus make optimization problems for obfuscation function generation efficient.
- ▹ We design a workflow with interactions between an untrusted server that performs computationally heavy tasks and a user device that performs tasks involving real location.
- ▹ We evaluate CORGI on samples from multiple regions selected from a real dataset (Gowalla - a social network based on user check-ins) to evaluate the impact of customization on utility, and privacy.

The rest of the paper is organized as follows. We introduce the CORGI framework and describe the key concepts used in our work in Section 2. In Section 3, we present the tree-based representation used in this work along with the policy model. In Section 4, we describe in detail the generation of the customizable and robust obfuscation function for each user. We present in Section 5, the architecture of our framework and detail the control flow on the user and server side. In Section 6, we evaluate our approach on a real dataset and compare it against a baseline. In Section 7 we go over the related work and we conclude the work by summarizing our contributions in Section 8.

2 PRELIMINARIES

In this section, we introduce the CORGI framework (Section 2.2) and the preliminaries (Section 2.1) of our geo-obfuscation approach.

2.1 Background

In this section, we formalize key concepts and notions for our proposed framework, introduced above.

Table 1: Main notations and their descriptions

Symbol	Description
v_i	Location i or node i
$\mathcal{V}$	Set of locations or nodes
p_{v_i}	Prior probability of location i
$d_{i,j}$	Distance between locations/nodes v_i and v_j
$\mathcal{V}^k$	The set of nodes with height k in the location tree
$\mathcal{T}^i$	A location tree with v_i as root node
$\mathbf{Z}^K$	Obfuscation Matrix at level K
$z_{i,j}$	Entry in the matrix at row i and column j
δ	Number of location nodes to be pruned
$\mathcal{S}$	Set of location nodes to be pruned

Obfuscation matrix. Generally, when considering the obfuscation range as a finite discrete location set $\mathcal{V} = \{v_1, ..., v_K\}$, an obfuscation function can be represented as a stochastic matrix $\mathbf{Z} = \left\{z_{i,j}\right\}_{K \times K}$ [21]. Here, each $z_{i,j}$ represents the probability of selecting $v_j \in \mathcal{V}$ as the obfuscated location given the real location $v_i \in \mathcal{V}$. For each real location v_i (corresponding to each row i of $\mathbf{Z}$), the *probability unit measure* needs to be satisfied:

$$\textstyle\sum_{j=1}^{K} z_{i,j} = 1, \forall i = 1, ..., K, \quad (1)$$

i.e., the sum probability of its obfuscated locations is equal to 1. In this paper, we consider the location set $\mathcal{V}$ at different granularity levels, and any real location can be only obfuscated to the locations at the same granularity level (details are introduced in Section 3.1).

Privacy Criteria. From the attacker's perspective, the user's actual and reported locations can be described as two random variables X and Y, respectively. We apply *Geo-Indistinguishability (Geo-Ind)* [2] as the privacy criterion for location privacy guarantees:

Definition 2.1. (ϵ-Geo-Ind) Given the obfuscation matrix $\mathbf{Z}$ that covers a set of locations $\mathcal{V}$ at the same granularity level, $\mathbf{Z}$ is called ϵ-Geo-Ind if and only if for each pair of real locations $v_i, v_j \in \mathcal{V}$ and any obfuscated $v_l \in \mathcal{V}$

$$\frac{\mathbb{P}\mathrm{r}\left(X = v_i \,|Y = v_l\right)}{\mathbb{P}\mathrm{r}\left(X = v_j \,|Y = v_l\right)} \le e^{\epsilon d_{i,j}} \frac{p_{v_i}}{p_{v_j}}, \quad (2)$$

where p_{v_i} and p_{v_j} denote the prior distributions of v_i and v_j, respectively, $\epsilon > 0$ is predetermined constant called privacy budget, and $d_{i,j}$ denotes the distance between v_i and v_j.

Equ. (2) indicates that the posterior of the user's location estimated from its obfuscated location is close to the user's prior location distribution and how close they are depends on the parameter ϵ. In other words, an adversary cannot obtain sufficient additional information from a user's obfuscated location.

The utility of our approach is measured based on the estimation error in traveling distance due to using obfuscated location in service provisioning. Given that user's real location is v_i, the

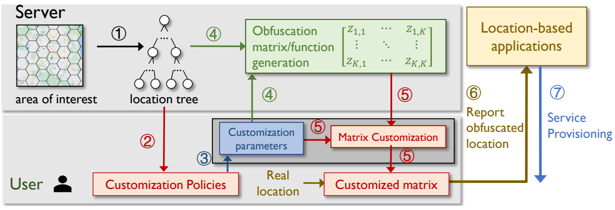

Server
①
④
Obfuscation matrix/function generation
$\begin{bmatrix} z_{1,1} & \cdots & z_{1,K} \\ \vdots & \ddots & \vdots \\ z_{K,1} & \cdots & z_{K,K} \end{bmatrix}$
Location-based applications
area of interest
location tree
④
⑤
⑥
Report obfuscated location
⑦
Service Provisioning
②
Customization parameters
⑤
Matrix Customization
③
⑤
User
Customization Policies
Real location
Customized matrix

Figure 1: Overview of CORGI framework.

obfuscated location generated is v_l, the target location is v_n, the utility is given by

$$U(v_i, v_l, v_n) =\mid d_{v_i,v_n} - d_{v_l,v_n} \mid . \tag{3}$$

where d_{v_i,v_n} can be implemented using any relevant distance function e.g., euclidean distance, haversine formula. If there are multiple target locations denoted by $v_1, \ldots, v_N$, the overall utility is computed as $\frac{1}{N}\sum_{n=1}^{N} U(v_i, v_j, v_n)$.

2.2 Framework

Our problem setting is that of *Location Based Services (LBS)* where users share their privatized locations with a server in order to receive service provisioning. This includes applications where reporting obfuscated location will directly affect QoS, such as taxi-hailing applications (e.g., Uber, Lyft), and other applications where QoS is not affected by obfuscated location, such as locality-based search engines (e.g., Yelp, Google Reviews), and citizen science applications (e.g., iNaturalist, eBird). There are three main actors in our setting: *users*, *third party providers*, and a *server*. *Users* wish to share their locations in a privacy-preserving manner with applications. They specify policies to state their customization preferences and have a privacy module/middleware running on their mobile device or on a trusted edge computer to assist with location hiding. *Third party providers* use the privatized locations shared by the user for providing services to the user. Finally, we have the *server*, which runs on the cloud with whom non-sensitive portions of the user preferences are shared and it takes care of computationally heavy operations. Note that, while an attacker can eavesdrop on the communication between the server and the user to obtain the location tree, users' requests (including privacy level, precision level, and the number of locations to be removed), and the generated obfuscation matrix. However, none of this information reveals anything about the user's exact location and does not weaken the Geo-Ind privacy guarantee of users' location reported by CORGI. Users do not trust either the third-party providers or the server with their sensitive location information or preferences. Figure 1 introduces the flow of CORGI and interactions among these three actors:
① The server generates a spatial index/location tree for an area of interest (Section 3.1).
② The location tree is shared with the users in the area to allow them to specify their preferences (Section 3.2).
③ CORGI evaluates the preferences on the user side to derive the customization parameters.
④ The server obtains the customization parameters and uses it to determine the privacy budget and generate the robust obfuscation function which guarantees Geo-Indistinguishability. The obfuscated function is represented by a set of probability distributions in an *obfuscation matrix* (Section 4.1).
⑤ Users receive the obfuscation function/matrix and customize it based on their needs (Section 4.3)[1].
⑥ ⑦ This customized obfuscation function is utilized to determine the user's obfuscated location, to be shared with location-based applications owned by third-party providers for the purpose of service provisioning.

3 MODELS

In this section, we introduce the models, including the location tree model (Section 3.1), i.e., how we organize locations at different granularity levels in a tree structure, and the user customization policies (Section 3.2), i.e., what attributes are considered in the customization.

3.1 Location Tree Model

We build a hierarchical index over a given spatial region for location representation. We design a tree-like structure, called *location tree*, where each level of the tree represents a particular granularity of location data, with finer levels of granularity represented at lower levels of the tree. This representation of locations is intuitive and makes it easier for users to specify the granularity of location sharing, that they are comfortable with.

In general, a tree can be represented by $\mathcal{T} = (\mathcal{V}, \prec)$, where $\mathcal{V}$ denotes the node set and $\prec$ describes the ordered relationship between nodes, i.e., $\forall v_i, v_j \in \mathcal{V}$, $v_j \prec v_i$ means that v_j is a child of v_i. $\forall v_i \in \mathcal{V}$, we let $\mathcal{N}\left(v_i\right)$ denote the set of v_i's children, i.e., $\mathcal{N}\left(v_i\right) = \left\{v_j \in \mathcal{V} \middle| v_j \prec v_i\right\}$. Here, we slightly overload the notation by letting v_i denote both location i and its corresponding node in the location tree. Given these notations, we formally define a location tree as follows:

Definition 3.1. (**Location Tree**) A location tree $\mathcal{T} = (\mathcal{V}, \prec)$ is a rooted tree, where

- ▷ the root node $v_r \in \mathcal{V}$ represents the whole area
- ▷ the tree is balanced and leaf nodes are $\left\{v_1, \ldots, v_K\right\}$;
- ▷ for each non-leaf node $v_i \in \mathcal{V}$, its children $v_j \in \mathcal{N}\left(v_i\right)$ represent a *partition* of v_i, i.e., locations in $\mathcal{N}\left(v_i\right)$ are disjoint and their union is v_i.

We partition the node set $\mathcal{V}$ in the location tree into H+1 levels: $\mathcal{V}^0, \ldots, \mathcal{V}^H$, where H is the height of the tree (i.e., the number of hops from the node to the deepest leaf). $\mathcal{V}^0$ represents the set of *leaf nodes*. We define *obfuscation in a location Tree* $(\mathcal{V}, \prec)$ as a function that maps a given real location $v_i \in \mathcal{V}^n$ to another location $v_k \in \mathcal{V}^n$ and both nodes are at the same level n.

We generate this location tree using Uber's H3[2] hexagonal hierarchical spatial index. For an area of interest, H3 takes as input the relevant longitude and latitude, along with resolution (between 0 and 15, with 0 being coarsest and 15 being finest) and outputs a hexagonal grid index (as illustrated in Figure 2). H3, then, partitions the area into contiguous hexagonal cells of the same size based on the resolution level. As H3 keeps the distance between the center point of a hexagon and its neighboring cells consistent, it is a better candidate for representing spatial relationships than grid-based systems such as Geohash. Figure 2 illustrates the location tree generated for Times Square, New York. Blue nodes at the highest granularity represent the leaf nodes. Red and green nodes at lower granularity represent the intermediate nodes. The root node encompasses the entire region.

[1] Matrix customization operations can be done on a trusted edge server if user device lacks the computational capability
[2] https://eng.uber.com/h3/

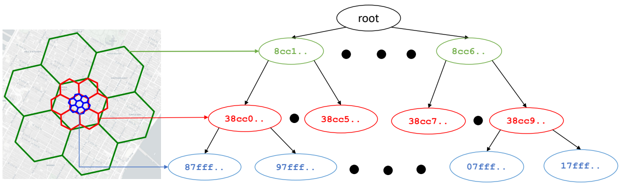


Figure 2: Location Tree of 3 levels generated using H3.

Nodes in each level do not overlap with each other. Our approach to location representation is inspired by previous works on spatial indexing such as R-Tree proposed by Beckmann et. al [5]. In Beckmann's approach, however, location nodes can overlap and are not disjoint partitions. In our cases, if any two location nodes overlap, it is hard to assess whether these two locations satisfy ϵ-*Geo-Ind* (Equation (2)) or not as a user can be in both of these nodes simultaneously.

3.2 User Customization Policies

Users express customization preferences by way of policies. These policies help determine the properties of the final obfuscation function that is generated. A policy captures users' customization requirements as follows:

$$< Privacy\_l, \;\; Precision\_l, \;\; User\_Preferences >$$

Privacy level or *Privacy_l* is a user-set parameter that determines the obfuscation range i.e., the set of locations/nodes from which users' obfuscated location is selected. Given a policy $\mathcal{P}$ with privacy_l = n, a *privacy forest* is the set of all sub-trees with nodes at level n as their root. Thus, the privacy forest contains all the possible locations that can be reported as obfuscated locations. As Figure 3 shows, if a user selects privacy level n, we first determine the nodes at height n ($\mathcal{V}^n$) which forms the privacy forest. Accordingly, a higher privacy level implies a wider range of obfuscated locations to select for users. In Figure 3, the red and blue colored subtrees indicate two different user policies both of which specify their privacy level as 2 but with the corresponding user at different real locations. For a particular user, if v is the ancestor of the user's real location at height n, the sub-tree with v as the root node includes all the locations that the user could report. The server can use *privacy level* to limit the number of locations in the obfuscation matrix for this user, and accordingly, reduce the overhead of generating it as well as improve the utility of location reporting compared to traditional approaches [21]. The *privacy level* also provides the flexibility for the user to specify the range of locations they are comfortable sharing. Note that, when a user selects a different privacy level, the obfuscation range is changed but the privacy budget ϵ remains the same, indicating that the same level of Geo-Ind (quantified by ϵ) is guaranteed no matter which privacy level is selected by the user.

Precision level (*Precision_l*) specifies the exact granularity at which the user reports their locations (e.g., neighborhood or block). For example, if a user requires the precision level to be 1, then their reported location/node is restricted to the set of nodes in level i.e., $\mathcal{V}^1$. Thus Precison_l gives users the flexibility to reduce the granularity at which location is shared depending on their needs. As the privacy level is the maximum possible granularity for location sharing, the precision level is always lower than the privacy level.

User Preferences specify users' preferred options for location selection and further narrows down the obfuscation range and therefore reduces the number of locations/nodes in the matrix. These may be expressed in a variety of ways, depending on the application at hand and the users' requests (e.g.black lists of locations, dynamic checks, etc). An intuitive approach is to encode preferences as Boolean predicates in the form $< var, \;\; op, \;\; val>$ where *var* denotes commonly used preferences for location sharing such as home, office, traffic, weather, driving_distance, etc; *op* is one among $\{=, \neq, <, >, \geq, \leq\}$ depending upon the variable; and *val* is assigned from the domain of the *var*.

An example of a policy modeled using these 3 attributes is as follows: *<privacy_l = 3, precision_l = 0, user_preferences = [popular = "True", distance* $\leq$ *5 miles*] This customization policy states that the user would prefer to have the *privacy forest* with nodes from level 3 (privacy_l = 3) and the nodes in this privacy forest represent their obfuscation range. From this set of possible locations, any of them which are not popular (determined using prior distribution), and has a distance higher than 5 miles from their real location should not be considered for reporting (user_preferences). Finally, when generating their obfuscated location, they would like it to be at the granularity of level 0 (precision_l = 0) i.e., the leaf nodes.

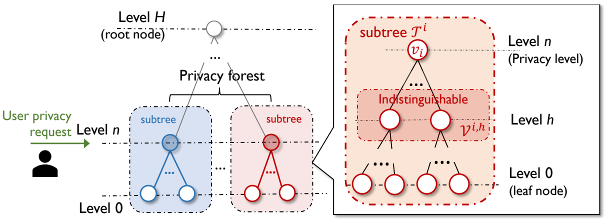
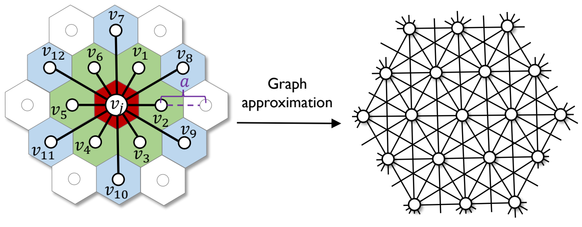


Figure 3: Tree-based geo-obfuscation.

4 GENERATING ROBUST OBFUSCATION MATRIX

We describe how to generate a robust obfuscation matrix, that preserves strong privacy guarantees while meeting users' customization policies, using the location tree. This is non-trivial, as introducing additional constraints based on policies affects the ability to obfuscate locations within certain regions and limits the range of possible obfuscated locations. In the rest of this section, we describe how to generate the obfuscation matrix based on the users' customization requirements $< Privacy\_l, \;\; Precision\_l,$ $User\_Preferences >$. In Section 4.1, we introduce how to generate the obfuscation matrix, based on the location tree, given users' requested privacy level $Privacy\_l$ using *linear programming (LP)* (formulated in Equ. (8)). As the number of Geo-I constraints in the LP is high, next, we present a graph approximation to reduce the number of the Geo-I constraints and improve the efficiency of the matrix generation (Section 4.2). We then customize the obfuscation matrix according to $User\_Preferences$ by removing locations based on user preferences (Section 4.3). Since removing locations might cause the Geo-I constraint violations in the matrix, we design how to generate a robust matrix to satisfy Geo-I constraints even after being customized (Section 4.4). Finally, we describe how to generate the matrix at the $Precision\_l$ requested by the user starting from the original matrix (Section 4.5).



(a) Build edge

(b) Graph

Figure 4: Graph approximation.

***(a) Build 12 edges connected to v_1: $e_{j,1},...,e_{j,12}$. The distance from $v_1,...,v_6$ to v_j is a; and the distance from $v_7,...,v_{12}$ to v_j is $\sqrt{3}a$.**

4.1 Feasibility Conditions for Geo-I

Given a user's requested privacy level n, the nodes at level n ($v_i \in \mathcal{V}^n$) and their children nodes ($\mathcal{N}\left(v_i\right)$) represent the possible set of obfuscated locations for a user. As the server does not know the user's real location or the subtree that contains user's real location, it has to generate an obfuscation matrix for each node v_i based on the leaf nodes in $\mathcal{N}\left(v_i\right)$. The server then returns all the generated obfuscation matrices to the user, and the user selects the obfuscation matrix according to their real location. Suppose users want to report a location with lower granularity than the leaf nodes for which the matrix is generated, they can do by applying precision reduction (discussed in Section 4.5) to generate the obfuscation matrix at the desired precision level.

Next, we introduce how to generate *feasible* obfuscation matrices of each subtree rooted at level n. Suppose that v_i is a node at level n, then we use $\mathcal{T}^i$ to denote the subtree rooted at v_i and let $\mathcal{V}^{i,0}$ denote the set of leaf nodes in $\mathcal{T}^i$, as shown in Fig. 3.

We use $\mathbf{Z}^0 = \left\{z_{k,l}\right\}_{|\mathcal{V}^{i,0}|\times|\mathcal{V}^{i,0}|}$ to represent the obfuscation matrix of $\mathcal{T}^i$ at precision level 0 (the highest precision level). We call $\mathbf{Z}^0$ is *feasible* if only if both ϵ-Geo-Ind (general case is defined in Equ. (2))

$$\frac{\mathbb{P}\mathrm{r}\left(X=v_j\,|Y=v_l\right)}{\mathbb{P}\mathrm{r}\left(X=v_k\,|Y=v_l\right)} \le e^{\epsilon d_{i,j}}\frac{p_{v_j}}{p_{v_k}}, \forall v_j, v_k, v_l \in \mathcal{V}^{i,0} \qquad (4)$$

and probability unit measure

$$\sum_{v_l\in\mathcal{V}^{i,0}} z_{k,l} = 1,\ \forall k \in \mathcal{V}^{i,0}, \qquad (5)$$

are satisfied. We let $Q = v_1, ..., v_M$ denote a set of places of interests which in our problem setting are locations where service is requested for e.g., passenger pickup. Given the target location $v_q \in Q$, the actual location v_k of a user, the obfuscated location v_l, the expected estimation error of moving distance caused by obfuscation matrix $\mathbf{Z}^0$ is obtained by

$$\Delta_q\left(\mathbf{Z}^0\right) = \sum_{v_k\in\mathcal{V}^{i,0}} \mathbb{P}\mathrm{r}\left(X=v_k\right) \sum_{v_l\in\mathcal{V}^{i,0}} z_{k,l}U(v_k, v_l, v_q). \qquad (6)$$

Given the probability distribution of target locations $\mathbb{P}\mathrm{r}\left(Q=v_q\right)$, then we can define the quality loss as the expected estimation error of moving distance as

$$\Delta\left(\mathbf{Z}^0\right) = \sum_{v_q\in\mathcal{V}^{i,0}} \mathbb{P}\mathrm{r}\left(Q=v_q\right)\Delta_n\left(\mathbf{Z}^0\right) \qquad (7)$$

$\mathbf{Z}^0$ is generated by solving the following linear programming (LP) problem

$$\min\ \Delta\left(\mathbf{Z}^0\right) \quad \text{s.t. Equ. (4) (5) are satisfied} \qquad (8)$$

i.e., minimize the expected estimation error of moving distance using the matrix $\mathbf{Z}^0$ to all the target locations. Once $\mathbf{Z}^0$ is generated, it will be delivered to the user and they are allowed to customize $\mathbf{Z}^0$ based on evaluation of *User_Preferences* (Section 4.3) and selection of the desired granularity level (Section 4.5).

While customizing, users select to remove a certain number of locations from the obfuscation range, and this results in a pruning of the matrix. Note that, after $\mathbf{Z}^0$ is pruned, the new matrix might no longer satisfy the Geo-I constraints in Equ. (4) (the details of matrix pruning will be introduced in Section 4.3). Intuitively, to avoid this potential privacy issue, we need to reserve more privacy budget when formulating the ϵ-Geo-Ind constraints and generate a more *robust* matrix that allows users to remove up to a certain number of locations in the matrix without violating Geo-I. The details of generating such a robust matrix will be given in Section 4.4.

4.2 Graph approximation to reduce the number of Geo-I constraints

According to the definition of Geo-Ind in Equ. (4), for each column (location) of the obfuscation matrix $\mathbf{Z}^0$, an ϵ-Geo-Ind constraint is generated for pairwise comparison of all locations, leading to a total of $O(K^3)$ constraints. This generates a very high computation load to derive $\mathbf{Z}^0$. To improve the time efficiency of the matrix calculation, we approximate the users' mobility on the 2D plane by a graph, where it is sufficient to enforce ϵ-Geo-Ind for each pair of neighboring nodes (*Theorem 4.2*), to enforce the ϵ-Geo-Ind constraints for all pairs of nodes. This reduces the number of constraints in LP from $O(K^3)$ to $O(12\times K^2) = O(K^2)$.

The method for approximating the hexagonal grid to a graph is illustrated in Figure 4. We connect each node v_i to not only the 6 immediate neighbors (denoted by $v_1, ..., v_6$) but also the 6 other diagonal neighbors (denoted by $v_7, ..., v_{12}$). We let a denote the distance between the immediate neighbors, computed based on the distance between their center points, and therefore the weight of each edge is set to a. Then, we can obtain a weighted graph $\mathcal{G}$ as Fig. 4(b) shows. The length of the shortest path between any pair of nodes v_j and v_k on the graph, denoted by $d_{\mathcal{G}}(v_j, v_k)$. Since the graph is undirected, we have $d_{\mathcal{G}}(v_j, v_k) = d_{\mathcal{G}}(v_k, v_j)$, $\forall v_j, v_k \in \mathcal{V}^{i,0}$. To ensure that ϵ-Geo-Ind on $\mathcal{G}$ to be a sufficient condition of the original ϵ-Geo-Ind constraint defined on the 2D plane, we need to guarantee that $d_{\mathcal{G}}(v_j, v_k)$ is no longer than their Euclidean distance $d_{j,k}$, i.e., $d_{\mathcal{G}}(v_j, v_k) \le d_{j,k}$ (the reason will be further explained in the proof of the *Theorem 4.2*). We first introduce Lemma 4.1 as preparation for Theorem 4.2.

Lemma 4.1. *$\forall v_j, v_k \in \mathcal{V}^{i,0}$, $d_{\mathcal{G}}(v_j, v_k) \le d_{j,k}$.*

Proof. We consider the two locations v_k and v_j on a polar coordinate system, where v_j is located at the origin point. We use $[r_k, \varphi_k]$ to represent v_k's polar coordinate, where $r_k \ge 0$ denotes the radial coordinate and $\varphi_k \in (-\pi, \pi]$ denotes the angular coordinate. As Fig. 5 shows, there are 6 different cases according to the value of φ_k: Case 1: $\varphi_k \in \left(-\frac{\pi}{6}, \frac{\pi}{6}\right]$, Case 2: $\varphi_k \in \left(\frac{\pi}{6}, \frac{\pi}{2}\right]$, Case 3: $\varphi_k \in \left(\frac{\pi}{2}, \frac{5\pi}{6}\right]$, Case 4: $\varphi_k \in \left(\frac{5\pi}{6}, -\frac{5\pi}{6}\right]$, Case 5: $\varphi_k \in \left(-\frac{5\pi}{6}, -\frac{\pi}{2}\right]$, and Case 6: $\varphi_k \in \left(-\frac{\pi}{2}, -\frac{\pi}{6}\right]$. In what follows, we prove that Lemma 4.1 is true in Case 1, where the conclusion can be applied to the other 5 cases due to the symmetricity of the 6 cases.

Case 1 can be further divided into two cases: Case 1(a) when $\varphi_k \in \left[0, \frac{\pi}{6}\right]$, and Case 1(b) when $\varphi_k \in \left[\frac{11\pi}{6}, 0\right]$.
In Case 1(a), we can always find a location v_1 that is u hops away from v_j in the direction of $\frac{\pi}{6}$ and w hops away from v_k in

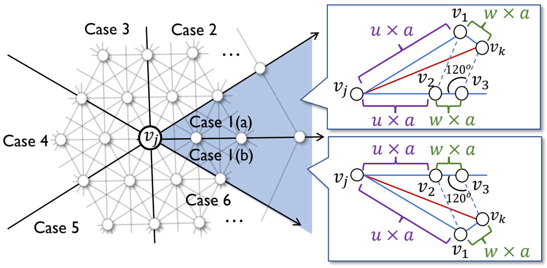
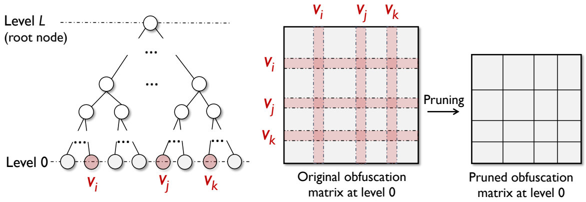


Figure 5: Proof of Lemma 4.1



Level L
(root node)
...
Level 0
v_i v_j v_k
v_i
v_j
v_k
Pruning
Original obfuscation matrix at level 0
Pruned obfuscation matrix at level 0

Figure 6: Matrix pruning (in the figure, v_i, v_j, v_k are the locations to be pruned).

the direction of $\frac{2\pi}{3}$. Starting from v_1 and v_k, if we move in the direction of $\frac{2\pi}{3}$, we can find a location v_2 and v_3 with the radial coordinate equal to 0. The number of hops from v_1 to v_j is equal to the number of hops from v_2 to v_j (u hops). The number of hops from v_k to v_1 is equal to the number of hops from v_3 to v_2 (w hops). Note that the length of each hop in the graph is a. In Case 1(b), we can always find a location v_1 that is u hops away from v_j in the direction of $-\frac{\pi}{6}$ and w hops away from v_k in the direction of $-\frac{\pi}{3}$. Similarly, we can find the corresponding v_2 that is u hops away from v_j and v_3 that is w hops away from v_2. In both Case 1(a)(b), according to the *Law of Sines*, we obtain that

$$d_{j,k} = \frac{\sin \angle v_j v_3 v_k}{\sin \angle v_j v_k v_3} d_{j,3} \geq d_{j,3} = (u+w)\,a \tag{9}$$

from which we can then derive that (according to the *triangle inequality* on a graph)

$$d_{\mathcal{G}}\left(v_j, v_k\right) \quad \leq \quad \underbrace{d_{\mathcal{G}}\left(v_j, v_1\right)}_{u\times a} + \underbrace{d_{\mathcal{G}}\left(v_1, v_k\right)}_{w\times a} \leq d_{j,k}$$

The proof is completed. □

Theorem 4.2. (Transitivity of ϵ-Geo-Ind) *To enforce ϵ-Geo-Ind for each pair of locations, it is sufficient to enforce ϵ-Geo-Ind only for each pair of neighboring peers in the graph $\mathcal{G}$.*

Proof. We pick up any pair of locations. Without loss of generality, we denote the two locations by (v_1, v_M) and denote their shortest path by $\mathcal{S}_{(v_1, v_M)} = ((v_1, v_2), ..., (v_{M-1}, v_M))$. We then prove that (v_1, v_M) satisfies ϵ-Geo-Ind if all the neighboring peers satisfy Geo-I.

Since $v_1, ..., v_M$ are in the shortest path from v_1 to v_M sequentially, $d_{1,M} \geq d_{\mathcal{G}}(1, M) = \sum_{l=1}^{M-1} d_{\mathcal{G}}(v_l, v_{l+1})$ (according to Lemma 4.1).

Because each neighboring peer $(v_{m_l}, v_{m_{l+1}})$ $(l = 1, ..., M-1)$ satisfies ϵ-Geo-Ind, for each obfuscated location v_k,

$$\begin{aligned}
& z_{1,k} - e^{\epsilon d_{1,M}} z_{M,k} && (10)\\
\leq \quad & z_{1,k} - e^{\epsilon \sum_{l=1}^{M-1} d_{\mathcal{G}}(v_l, v_{l+1})} z_{M,k} && (11)\\
= \quad & \sum_{l=1}^{M-1} \underbrace{\left(z_{l,k} - e^{\epsilon d_{l,l+1}} z_{l+1,k}\right)}_{\leq 0 \text{ since } (v_l, v_{l+1}) \text{ satisfy } \epsilon\text{-Geo-Ind}} \quad e^{\epsilon \sum_{h=1}^{l-1} d_{h,h+1}} && (12)\\
\leq \quad & 0, && (13)
\end{aligned}$$

indicating that (v_1, v_M) satisfy ϵ-Geo-Ind. The proof is completed. □

Note that enforcing ϵ-Geo-Ind for neighbors in $\mathcal{G}$ provides a sufficient condition for the original ϵ-Geo-Ind constraints (defined in Equ. (2)), but not a necessary condition, which means it might shrink the feasible region of the original LP defined in Equ. (8), leading to a higher quality loss ($\Delta\left(\mathbf{Z}^0\right)$).

4.3 Customization by Matrix Pruning

After receiving obfuscation matrices from the server, the user can select the matrix $\mathbf{Z}^0$ based on their real location and can customize the matrix by removing the locations that do not satisfy their preferences. For example, in Figure 6, the three nodes marked in red at Level 0, $\{v_i, v_j, v_k\}$, are to be pruned[3]. The 3 corresponding rows and columns in the matrix $\mathbf{Z}^0$ are highlighted and in the next step, they are removed.

The resulting matrix $\mathbf{Z}^0_*$ is considered *feasible*, only if it still satisfies the probability unit measure for each row in the matrix as per Equ. (1). We denote the set of nodes (that do not satisfy the user's preferences) to be removed from the matrix by $\mathcal{S}$ ($\mathcal{S} \subseteq \mathcal{V}^0$). After pruning, the new obfuscation matrix $\mathbf{Z}^0_*$ is of dimensions $m \times m$ where $m = |\mathcal{V}^0 - \mathcal{S}|$. This process called *matrix pruning* is carried out as follows:

- ▹ Remove the rows and the columns of nodes with indices in $\mathcal{S}$ from $\mathbf{Z}^0$ to create $\mathbf{Z}^0_*$.
- ▹ For each remaining row i in $\mathbf{Z}^0_*$, multiply each entry in the matrix $z_{i,k}$ by $\frac{1}{1-\sum_{l\in\mathcal{S}} z_{i,l}}$, i.e., $z_{i,k} \leftarrow \frac{z_{i,k}}{1-\sum_{l\in\mathcal{S}} z_{i,l}}$.

This ensures that the entries in each row still satisfy the probability unit measure, i.e.,

$$\begin{aligned}
\sum_{k\in\mathcal{V}^0\setminus\mathcal{S}} z_{i,k} \quad &= \quad \frac{\sum_{k\in\mathcal{V}^0\setminus\mathcal{S}} z_{i,k}}{1-\sum_{l\in\mathcal{S}} z_{i,l}} && (14)\\
&= \quad \frac{\sum_{k\in\mathcal{V}^0} z_{i,k} - \sum_{k\in\mathcal{S}} z_{i,k}}{1-\sum_{l\in\mathcal{S}} z_{i,l}} && (15)\\
&= \quad 1.
\end{aligned}$$

4.4 Ensuring Robustness of Customized Matrix

After matrix pruning, although the pruned matrix satisfies the probability unit measure, it might not satisfy ϵ-Geo-I since in each column k, the entries $z_{i,k}$ $(i = 1, ..., K)$ are multiplied by different factors $\frac{1}{1-\sum_{l\in\mathcal{S}} z_{i,l}}$. We denote the size of the set of nodes to be pruned from the matrix as δ i.e., δ=$|\mathcal{S}|$, and define *δ-prunable* robust matrix as follows:

Definition 4.3. An obfuscation matrix $\mathbf{Z}$ is called *δ-prunable* if, after removing up to δ number of nodes from $\mathbf{Z}$ through *matrix*

[3] Even though pruning can be done at any level of the tree, it makes the most sense to do it for locations at leaf node (at highest granularity) so as to remove only the exact locations and avoid over-pruning.

pruning, the new matrix $\mathbf{Z}_*$ still satisfies ϵ-Geo-Ind, i.e., $\forall i, j, k$,

$$\frac{z_{i,k}}{1-\sum_{l\in\mathcal{S}} z_{i,l}} - e^{\epsilon d_{i,j}} \frac{z_{j,k}}{1-\sum_{l\in\mathcal{S}} z_{j,l}} \le 0, \forall \mathcal{S} \subseteq \mathcal{V}^{i,0} \text{ s.t. } |\mathcal{S}| \le \delta \tag{16}$$

In order to make an obfuscation matrix δ-prunable, we need to reserve more privacy budget $\epsilon_{i,j}$ (defined in Equ. (17)) for each pair of locations v_i and v_j, such that even a certain number of locations are pruned from the matrix, the Geo-I constraints of v_i and v_j are still satisfied. We now define *reserved privacy budget*, denoted by $\epsilon_{i,j}$, as follows.

Definition 4.4. The reserved privacy budget $\epsilon_{i,j}$ for each pair of locations v_i and v_j where i, j are their indices in the obfuscation matrix is given by,

$$\epsilon_{i,j} = \frac{1}{d_{i,j}} \ln\left(\max_{\mathcal{S}\subseteq\mathcal{V}^{i,0} \text{ s.t. } |\mathcal{S}|\le\delta} \frac{1-\sum_{l\in\mathcal{S}} z_{j,l}}{1-\sum_{l\in\mathcal{S}} z_{i,l}} \right) \tag{17}$$

Proposition 4.5. *A sufficient condition for* $\mathbf{Z}$ *to be* δ*-prunable is to satisfy*

$$z_{i,k} - e^{\left(\epsilon-\epsilon_{i,j}\right)d_{i,j}} z_{j,k} \le 0, \forall i, j, k. \tag{18}$$

Proof. Given that Equation (18) is satisfied, then for each column k, $\forall i, j, \mathcal{V}_0' \in \mathcal{V}_0$ with $|\mathcal{V}_0'| \le \delta$,

$$\begin{aligned}
& \frac{z_{i,k}}{1-\sum_{l\in\mathcal{V}_0'} z_{i,l}} - e^{\epsilon d_{i,j}} \frac{z_{j,k}}{1-\sum_{l\in\mathcal{V}_0'} z_{j,l}} \\
= & \frac{1}{1-\sum_{l\in\mathcal{V}_0'} z_{j,l}} \left(\frac{1-\sum_{l\in\mathcal{V}_0'} z_{j,l}}{1-\sum_{l\in\mathcal{V}_0'} z_{i,l}} z_{i,k} - e^{\epsilon d_{i,j}} z_{j,k} \right) \\
\le & \frac{1}{1-\sum_{l\in\mathcal{V}_0'} z_{j,l}} \left(e^{\epsilon_{i,j} d_{i,j}} z_{i,k} - e^{\epsilon d_{i,j}} z_{j,k} \right) \\
= & \frac{e^{\epsilon_{i,j} d_{i,j}}}{1-\sum_{l\in\mathcal{V}_0'} z_{j,l}} \underbrace{\left(z_{i,k} - e^{\left(\epsilon-\epsilon_{i,j}\right)d_{i,j}} z_{j,k} \right)}_{\le 0 \text{ according to Equ. (18)}} \le 0.
\end{aligned}$$

□

Thus, we can state the minimization problem for *robust matrix generation*, min $\Delta\left(\mathbf{Z}^0\right)$, where the objective function (Equ. (7)) and equality constraints remains the same as earlier (Equ. (5)) but the inequality constraints are updated to Equ. (18) using reserved privacy budget.

In order to calculate $\epsilon_{i,j}$ in Equ. (17), we need to consider all the possible subsets of $\mathcal{S} \subseteq \mathcal{V}^{i,0}$ with the cardinality no larger than δ. The complexity of computing the reserved privacy budget increases exponentially with δ. Therefore, we define an approximation of $\epsilon_{i,j}$, denoted by $\epsilon_{i,j}'$ as follows:

$$\epsilon_{i,j}' = \frac{1}{d_{i,j}} \ln\left(\frac{1 - \frac{\max_{\mathcal{S}\subseteq\mathcal{V}^{i,0} \text{ s.t. } |\mathcal{S}|\le\delta} \sum_{l\in\mathcal{S}} z_{j,l}}{e^{\epsilon d_{i,j}}}}{1 - \max_{\mathcal{S}\subseteq\mathcal{V}^{i,0} \text{ s.t. } |\mathcal{S}|\le\delta} \sum_{l\in\mathcal{S}} z_{j,l}} \right) \tag{19}$$

Proposition 4.6. *The matrix generated by replacing* $\epsilon_{i,j}$ *with* $\epsilon_{i,j}'$ *in Equ. (18) is an upper bound of the solution.*

Proof.

$$\begin{aligned}
& \epsilon_{i,j} \\
= & \frac{1}{d_{i,j}} \ln\left(\max_{\mathcal{S}\subseteq\mathcal{V}^{i,0} \text{ s.t. } |\mathcal{S}|\le\delta} \frac{1-\sum_{l\in\mathcal{S}} z_{j,l}}{1-\sum_{l\in\mathcal{S}} z_{i,l}} \right) \\
\le & \frac{1}{d_{i,j}} \ln\left(\max_{\mathcal{S}\subseteq\mathcal{V}^{i,0} \text{ s.t. } |\mathcal{S}|\le\delta} \frac{1 - \frac{\sum_{l\in\mathcal{S}} z_{i,l}}{e^{\epsilon d_{i,j}}}}{1-\sum_{l\in\mathcal{S}} z_{i,l}} \right) \quad (\text{as } e^{\epsilon d_{i,j}} z_{j,l} \le z_{i,l}) \\
\le & \frac{1}{d_{i,j}} \ln\left(\frac{1 - \frac{\max_{\mathcal{S}\subseteq\mathcal{V}^{i,0} \text{ s.t. } |\mathcal{S}|\le\delta} \sum_{l\in\mathcal{S}} z_{i,l}}{e^{\epsilon d_{i,j}}}}{1 - \max_{\mathcal{S}\subseteq\mathcal{V}^{i,0} \text{ s.t. } |\mathcal{S}|\le\delta} \sum_{l\in\mathcal{S}} z_{i,l}} \right) = \epsilon_{i,j}'
\end{aligned}$$

□

To calculate $\epsilon_{i,j}'$, we need to find the top δ number of $z_{j,l}$ with $v_l \in \mathcal{V}^{i,0}$, which takes $O(K \log K)$ in the worst case. According to Proposition 4.6, by replacing $\epsilon_{i,j}$ with $\epsilon_{i,j}'$ in Equ. (18), we can obtain a sufficient condition of Equ. (18).

$$z_{i,k} - e^{\left(\epsilon-\epsilon_{i,j}'\right)d_{i,j}} z_{j,k} \le 0, \forall i, j, k. \tag{20}$$

By replacing Equ. (18) with this sufficient condition expressed in Equ. (19), we have the robust matrix generation problem which is an upper bound on the solution.

$$\min\ \Delta\left(\mathbf{Z}^0\right) \quad \text{s.t. Equ. (20) (5) are satisfied} \tag{21}$$

Algorithm 1: Robust matrix generation

```
1  Function generateRobustMatrix($\mathcal{V}$, $Prob^0$, $\delta$, $\epsilon$, $t$):
2      $i = 0$
3      $\mathbf{Z}_i[0,0] \ldots [|\mathcal{V}|-1, |\mathcal{V}|-1] = 0$
4      $\mathbf{Z}_i$ = LPSolver($\mathcal{V}$, $\epsilon$, $Prob^0$)
5                      ▷ Matrix generated by solving Equ. (8)
6      $RPB[0,0] \ldots [|\mathcal{V}|-1, |\mathcal{V}|-1] = 0$
7      do
8          $i$ +=1
9          $RPB$ = computeRPB($\mathcal{V}$, $\mathbf{Z}_i$, $\delta$)
10          ▷ Reserved Privacy Budget (RPB) using Equ. (19)
11         $\mathbf{Z}_i$ = LPSolver($\mathcal{V}$, $\epsilon$, $Prob^0$, $RPB$)
12                     ▷ Matrix generated by solving Equ. (21)
13     while $i \le t$
14     return $\mathbf{Z}_t$
```

Algorithm 1 takes as input the set of nodes $\mathcal{V}$, their prior probability distribution $Prob^0$, number of locations to be pruned δ, privacy parameter ϵ, and the number of iterations for convergence t (which is determined empirically based on convergence experiments, see Section 6). The non-robust matrix is generated first by solving the linear programming problem expressed in Equ. (8) (Step 4). For storing the Reserved Privacy Budget (RPB) for each pair of locations v_i and v_j, we initialize a matrix denoted by RPB (Step 6). We iteratively compute the RPB matrix using Equ. (19) and then use it to generate the matrix using the linear programming problem expressed in Equ. (21). This process is repeated for t iterations until the RPB matrix, as well as the matrix generated using it, converges. The robust obfuscation matrix is returned in the final step.

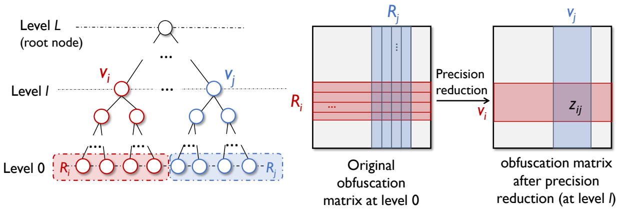


Figure 7: Matrix precision reduction.

4.5 Matrix Precision Reduction

In Figure 7, the original obfuscation matrix is generated for level 0, i.e., the set of leaf nodes. Suppose the user specifies a value l as its Precision_l. *Matrix precision reduction* generates the obfuscation matrix at level l, $\mathbf{Z}^l$ ($l > 0$), given the obfuscation matrix at level 0, $\mathbf{Z}^0$. As illustrated in the figure, the new matrix is generated by replacing all the rows of the descendant leaf nodes with their corresponding ancestor nodes at level l. For each pair of nodes v_i and v_j at level l, we use $\mathcal{N}(v_i)$ and $\mathcal{N}\left(v_j\right)$ to represent the set of their descendant leaf nodes, respectively. The probability of selecting v_j as the obfuscated location given the real location v_i is calculated using Bayes' theorem.

$$z^l_{i,j} = \frac{\sum_{v_m \in \mathcal{N}(v_i)} p_{v_m} \sum_{v_n \in \mathcal{N}\left(v_j\right)} z^0_{m,n}}{p_{v_i}} \tag{22}$$

where p_{v_m} and p_{v_i} denote the prior distributions of v_m and v_i respectively. Note that, $p_{v_i} = \sum_{v_m \in \mathcal{N}(v_i)} p_{v_m}$.

Proposition 4.7. *Matrix precision reduction preserves both probability unit measure and ϵ-Geo-Ind.*

Proof. First, we check the probability unit measure.

$$\begin{aligned} &\mathcal{V}_0 = \cup_{s_j \in \mathcal{V}_l} \mathcal{R}_j \\ \Rightarrow &\sum_{s_j \in \mathcal{V}_l} \sum_{s_v \in \mathcal{R}_j} z^0_{u,v} = \sum_{s_v \in \mathcal{V}_0} z^0_{u,v} = 1. \end{aligned} \tag{23}$$

We take sum of the entries in each row i in $\mathbf{Z}^l$,

$$\begin{aligned} \sum_{s_j \in \mathcal{V}_l} z^l_{i,j} &= \sum_{s_j \in \mathcal{V}_l} \frac{\sum_{s_u \in \mathcal{R}_i} p_u \sum_{s_v \in \mathcal{R}_j} z^0_{u,v}}{p_i} \\ &= \frac{\sum_{s_u \in \mathcal{R}_i} p_u \left(\sum_{s_j \in \mathcal{V}_l} \sum_{s_v \in \mathcal{R}_j} z^0_{u,v}\right)}{p_i} \\ &= \frac{\sum_{s_u \in \mathcal{R}_i} p_u}{p_i} = \frac{p_i}{p_i} = 1, \end{aligned}$$

i.e., each row i satisfies the probability unit measure.

We then check ϵ-Geo-Ind for each column k in $\mathbf{Z}^l$: $\forall s_i, s_j$

$$\begin{aligned} &z^l_{i,k} - e^\epsilon z^l_{j,k} \\ = &\frac{\sum_{s_u \in \mathcal{R}_i} \sum_{s_w \in \mathcal{R}_k} p_u z^0_{u,w}}{p_i} - e^\epsilon \frac{\sum_{s_v \in \mathcal{R}_j} \sum_{s_w \in \mathcal{R}_k} p_v z^0_{v,w}}{p_j} \\ = &\sum_{s_w \in \mathcal{R}_k} \left(\frac{\sum_{s_v \in \mathcal{R}_j} \sum_{s_u \in \mathcal{R}_i} p_u p_v \left(z^0_{u,w} - e^\epsilon z^0_{v,w}\right)}{p_i p_j} \right) \leq 0 \end{aligned}$$

since $z^0_{u,w} - e^\epsilon z^0_{v,w} \leq 0 \ \forall u, v, w$. □

Algorithm 2 presents the approach for matrix precision reduction given the matrix for leaf nodes ($\mathbf{Z}^0$), the location tree ($\mathcal{T}$), the prior probability distribution of the leaf nodes $prob^0$, and the

Algorithm 2: Precision Reduction Function

```
    Input: Obfuscation matrix (at level 0) $\mathbf{Z}^0$, Location Tree $\mathcal{T}$,
           Precision Level $l$
    Output: Obfuscation Matrix (at level l) $\mathbf{Z}^l$
1   Function precisionReduction($\mathbf{Z}^0, \mathcal{T}, l$):
2       $\mathcal{V}^l$ = getNodes($\mathcal{T}, l$)     ▷ Get nodes at precision level
3       $\mathbf{Z}^l[0,0] \ldots [|\mathcal{V}^l| - 1, |\mathcal{V}^l| - 1] = 0$
4       for $i \in 0, \ldots, |\mathcal{V}^l| - 1$ do
5           for $j \in 0, \ldots, |\mathcal{V}^l| - 1$ do
6               $num = 0, den = 0$
7               for $u \in 0 \ldots |\mathcal{N}(v_i)| - 1$ do
8                   $row\_sum = 0$
9                   for $v \in 0 \ldots |\mathcal{N}\left(v_j\right)| - 1$ do
10                      $row\_sum = row\_sum + z^0_{u,v}$
11                  end
12                  $num = num + p_{\mathcal{V}^0}[u] \times row\_sum$
13                  $den = den + p_{\mathcal{V}^0}[u]$
14              end
15              $z^l_{i,j} = \frac{num}{den}$
16          end
17      end
18      return $\mathbf{Z}^l$
```

precision level (l) which specifies the granularity/height of the tree of the reported location. First, we get the set of nodes ($\mathcal{V}^l$) from level l (Step 2). We initialize the new obfuscation matrix with dimensions based on the set of nodes retrieved. For each pair of location nodes (v_i, v_j) in $\mathcal{V}^l$, we compute their corresponding probability in the new matrix ($z^l_{i,j}$) by using the probabilities of their leaf nodes, $\mathcal{N}(v_i)$ and $\mathcal{N}\left(v_j\right)$ respectively, in Equ. (22) (Steps 4-16). The prior probability distribution for the leaf nodes $p_{\mathcal{V}^0}$ in this subtree can be obtained by querying the server[4]. Finally, the new matrix $\mathbf{Z}^l$ for level l is returned. Thus using matrix precision reduction, we are able to save the overhead of generating the obfuscation matrix when the user chooses to share at a lower granularity than the leaf nodes.

5 LOCATION OBFUSCATION BY CORGI

In this section, we describe the steps performed CORGI in order to generate the obfuscated location of a user. This process is sketched in Figure 8 and we explain the steps on the user and server side in detail below.

5.1 Server side

CORGI on server side takes as input the Privacy level *Privacy_l* and the number of locations to be pruned (δ). Algorithm 3 describes the steps for determining the obfuscation range (represented by the privacy forest) and generating the obfuscation matrix. First, the server determines the nodes at the given privacy level $\mathcal{V}^l$ by performing a Breadth First Search in the Location Tree $\mathcal{T}$(Step 2). Second, it initializes the privacy forest as a dictionary where the key is a subtree and the value is the obfuscation

[4] We assume that the prior probability distribution is readily available based on publicly available information. We explain how it is computed for a real dataset in Section 6. We disregard communication and computation cost for this as it is a relatively small vector and only has to be periodically updated.

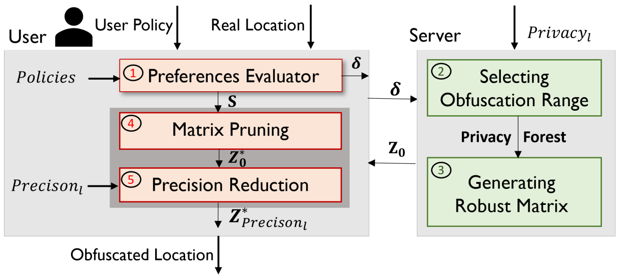


Figure 8: Steps in generating the obfuscated location

matrix for the leaf nodes of that subtree (Step 3). The system then iterates through each node v_i at the privacy level and generates an obfuscation matrix for each of them. For this purpose, the server has to determine the subtree rooted at v_i and perform a Depth First Search to determine the leaf nodes of that subtree (Steps 5-6). Next, it calls the *generateRobustMatrix* (Algorithm 1) with the set of leaf nodes and the number of locations to be pruned (Step 7) [5]. The robust obfuscation matrix $\mathbf{Z}^0$ thus generated for the leaf nodes is added to the dictionary with the subtree as the key (Steps 8-9). After iterating through all the nodes at the set privacy level and generating obfuscation matrices for each of their descendant leaf nodes, the final privacy forest *PF* is returned.

Algorithm 3: Generate Robust Obfuscation Matrices based on Obsfucation Range

```
   Input: Location Tree $\mathcal{T}$, Privacy Level $l$, Prune Parameter $\delta$
   Output: Privacy Forest $PF$
1  Function generateMatrix($\mathcal{T}, l, \delta$):
2      $\mathcal{V}^l$ = getNodes($\mathcal{T}, l$)
3      $PF = \{\}$
4      for $v_i \in \mathcal{V}^l$ do
5          $\mathcal{T}^i$ = findSubTree($v_i, \mathcal{T}, l$)
6          $\mathcal{V}^0$ = getNodes($\mathcal{T}^i$, 0)
7          $Prob^0$ = getPriorProbDist($\mathcal{V}^0$)
8          $\mathbf{Z}^0$ = generateRobustMatrix($\mathcal{V}^0, Prob^0, \delta, \epsilon, t$)
9          $PF[\mathcal{T}^i] = \mathbf{Z}^0$
10     end
11     return $PF$
```

5.2 User side

Users input their policies as well as their real location in order to generate the obfuscated location. First, CORGI on the user side determines the subtree $\mathcal{T}^i$ that contains the user's real location and is rooted at $\mathcal{P}$.Privacy_l (Step 1). We slightly abuse the notation here as v_i denotes the real location as well as the node in the tree that contains the actual user's location. Next, the user preferences are evaluated on the leaf nodes of that subtree and determine the set of nodes ($\mathcal{S}$) that are to be pruned. (Step 2 of Fig 8). The number of locations in this set ($|\mathcal{S}|$) along with $\mathcal{P}$.Privacy_l is passed to the server side (Step 4). From the privacy forest returned by the server (based on Algorithm 3), the obfuscation matrix $\mathbf{Z}^0$ corresponding to the subtree that contains the user's real location is selected (Step 5). Next, the system prunes this matrix by calling the Matrix Pruning Algorithm (*pruneMatrix*) with the set of nodes to be pruned (Step 6). The pruned matrix $\mathbf{Z}^0_*$ is updated to reflect the required granularity specified in $\mathcal{P}$.Precision_l (Step 7). From this final matrix, the row corresponding to the node at $\mathcal{P}$.Precision_l, which contains the ancestor of the real location of the user is selected. The obfuscated location v_j is selected from the row by sampling based on the probability distribution (Step 8).

5.3 Discussion

It is possible that when evaluating user preferences at the time of location sharing more than δ locations need to be pruned based on the user preferences. In such a situation, there are two options for customization: (1) Satisfy all the user preferences which result in a set of locations to be pruned $\mathcal{S}$ where $|\mathcal{S}| > \delta$ which leads to Geo-Ind violation, (2) Satisfy some of the policies in $\mathcal{P}$ such that $|\mathcal{S}| \leq \delta$ locations which leads to *policy violations* (there exists a location $v \in \mathbf{Z}$ such that it does not satisfy a policy in $\mathcal{P}$). Both these violations may occur if based on the policies a large number of locations have to be pruned from the matrix i.e., $|\mathcal{S}|$ is large. In such a case, CORGI finds it impossible to meet the δ requested by the user as well as generate an obfuscation matrix that is robust.

Algorithm 4: Generate Obfuscated Location

```
   Input: Location Tree $\mathcal{T}$, Real Location $v_i$, Policy $\mathcal{P}$
   Output: Obfuscated Location $v_j$
1  Function generateObsfucatedLocation($v_i, \mathcal{P}$):
2      $\mathcal{T}^i$ = findSubTree($v_i, \mathcal{T}, \mathcal{P}$.Privacy_l)
3      $\mathcal{S}$ = eval($\mathcal{T}^i, \mathcal{P}$.User_Preferences)
4      $PF$ = generateMatrix($\mathcal{T}, \mathcal{P}$.Privacy_l, $|\mathcal{S}|$)
5      $\mathbf{Z}^0 = PF[\mathcal{T}^i]$
6      $\mathbf{Z}^*_0$ = pruneMatrix( $\mathbf{Z}^0, \mathcal{S}$)
7      $\mathbf{Z}^*_l$ = precisionReduction($\mathbf{Z}^*_0, \mathcal{T}^i, \mathcal{P}$.Precision_l)
8      $v_j$ = sample($\mathbf{Z}^*_l$[ancestor($v_i, \mathcal{P}$.Precision_l)])
9      return $v_j$
```

In this work, we have used approximation techniques in order to reduce the number of constraints (see Section 4.2). An alternative method is to incorporate optimization decomposition in the linear programming model itself (similar to [20]) which would lead to improvement in utility. Currently, CORGI supports point queries and does not handle trajectory data. An adversary who has knowledge of the local traffic flows can eliminate impossible locations from the obfuscation range considering the restrictions of vehicle traffic flow. This can be extended by replacing the privacy notion of Geo-Indistinguishability with Trajectory Indistinguishability [22]. Trajectory-Ind guarantees that an adversary is unable to distinguish the vehicle's real trajectory from the other fake trajectories generated by our approach (TrafficAdaptor). This can be currently implemented in CORGI by using customization parameters to limit the obfuscation range to contain only the locations that are reachable based on current traffic flow and/or along the fake trajectory generated by TrafficAdaptor. Local Differential Privacy (LDP) has recently emerged as an approach to avoid using a centralized server and perturb users' data locally before it leaves their device [15]. However, to the best of our knowledge, most previous works that utilize LDP have mainly

[5] The privacy parameter (ϵ) and the number of iterations to convergence when generating the robust matrix (t) are set universally for all the users. ϵ is updated based on the user's customization needs.

focused on releasing population statistics [30] and not location privacy as presented in this paper.

6 EXPERIMENTS

6.1 Experimental setup

Datasets: We use the Gowalla dataset [8] for our experiments. Gowalla is a location-based social networking website where users share their location check-in data. The dataset includes the following attributes: *[user, check-in time, latitude, longitude, location id]*. We sampled the user check-ins from two different regions in the Gowalla dataset: 1) San Francisco (D1), and New York (D2). We choose these two regions because they both had a dense distribution of check-ins distributed over a large area. The first sample (D1) includes 38,523 check-ins and the second sample (D2) includes 13,384 check-ins. We use D1 as the default region in the simulation. For both samples, we generated the root node which covers the entire region at resolution 6 followed by the children for this root node at resolution 7. We repeated the process two more times and generated a tree of height 3 with 343 leaf nodes. For generating customization preferences, we analyzed the sample and came up with heuristics to identify a user's home, office, and their outlier locations (where the user visited rarely and at odd times). We also analyzed the number of check-ins per location in order to identify what locations are popular and at what times. Using this metadata we generated realistic user preferences such as *home = "False", outlier = "False", popular = "True"* .

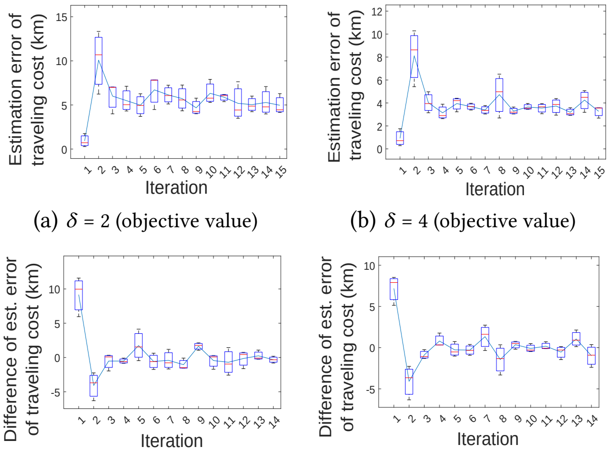
Priors: We computed prior probability for the leaf nodes in the location tree by counting number of user check-ins within that node. For intermediate nodes (higher up in the tree), the prior was computed by aggregating the priors of its children nodes.

Baseline: We used the commonly used mechanism of *linear programming (LP)* for implementing the baseline [6, 21, 28]. The LP method used in these approaches, sets their objective as to minimize quality loss like CORGI, while only satisfying Geo-Ind defined in Equ. (4). Hence, their generated matrix is likely to violate Geo-Ind constraints, if some locations are removed by users. The matrix generated by CORGI satisfies not only Equ. (4), but also the stronger constraints defined in Equ. (16) that reserves extra privacy budget $\epsilon_{i,j}$ (defined in Equ. (17)) to ensure Geo-Ind is still satisfied after user customization.

Implementation: All of the algorithms were implemented using the state-of-the-art Linear Programming tool kit in Matlab. The full implementation including scripts to run the experiments is available on Github[6]. In addition to the prototype, we have also developed a web application of CORGI where users can observe in the real-time impact of various customization parameters on privacy and utility [18].

6.2 Experimental results

6.2.1 Convergence. In this experiment, we test the convergence of the *quality loss* of CORGI, measured as the mean estimation error of traveling distance (implemented using *Haversine distance*) to all the target locations. We set *NR_TARGET* = 49 (number of target locations that are randomly selected from a list of leaf nodes), ϵ = 15 km^{-1} and used the priors generated from the Gowalla dataset. We ran two sets of experiments: when $\delta = 2$ and $\delta = 4$, where δ is the number of locations that the user wishes to remove after customization. In each group, we ran the experiment for 10 times, and depicted the convergence of the quality loss in Fig. 9(a)(c) (when $\delta = 2$) and Fig. 9(b)(d) (when $\delta = 4$). In all four figures, the x-axis denotes the iteration index. In Fig. 9(a)(b), the y-axis represents the quality loss, while in Fig. 9(c)(d), the y-axis represents the difference between quality loss in consecutive iterations. Here, a lower value on the y-axis denotes better convergence as there is little difference between entries in the matrix after each round. As illustrated in Figure 9(a)(b)(c)(d), the differences between quality loss in consecutive iterations converges in approximately 4 iterations for both values of δ. We conservatively terminate the program after 10 iterations for rest of the experiments.



(a) δ = 2 (objective value)

(b) δ = 4 (objective value)

(c) δ = 2 (difference of the objective value in consecutive iterations)

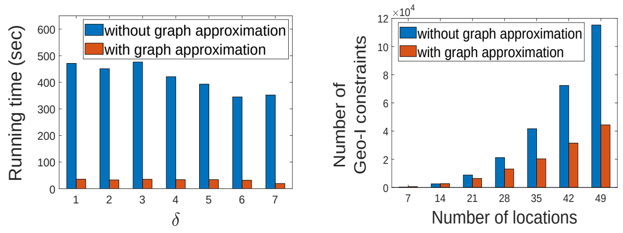
(d) δ = 4 (difference of the objective value in consecutive iterations)

Figure 9: Convergence of the objective value (estimation error of traveling costs).

6.2.2 Computation time of the obfuscation matrix generation. CORGI uses graph approximation to improve the time-efficiency of the obfuscation matrix generation (Section 4.2). In this experiment, we evaluate how much computation time is reduced by the graph approximation. Fig. 10(a) compares the computation time with and without graph approximation, with δ increased from 1 to 7. Fig. 10(a) demonstrates that the graph approximation has reduced the running time by 92.34% on average. The graph approximation improves the time efficiency of the matrix generation significantly since it reduces the number of Geo-Ind constraints from $O(K^3)$ to $O(K^2)$. Fig. 10(b) compares the number of Geo-Ind constraints without and with graph approximation, with the number of locations increasing from 7 to 49. The figure shows that the number of Geo-Ind constraints is reduced by 54.58% on average.

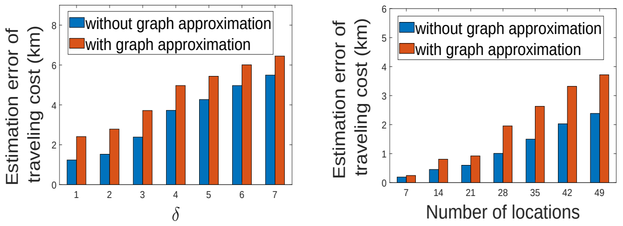
Fig. 11(a)(b)) compares the quality loss with and without graph approximation, with δ increased from 1 to 7 and the number of locations increased from 7 to 49, respectively. As expected, when using graph approximation, the quality loss is higher as it shrinks the feasible region of the LP defined in Equ. (8) which has been analyzed in Section 4.2. Note that, graph approximation is optional and can be turned off in CORGI for better utility if the running time is not a concern.

[6] https://github.com/User-Privacy/CORGI



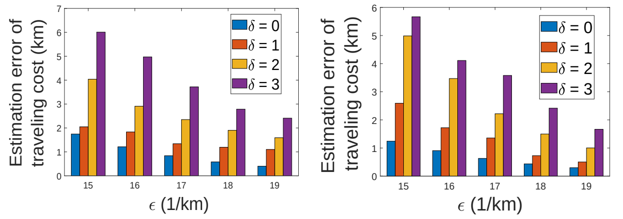
(a) Running time (b) Number of Geo-Ind constraints

Figure 10: Efficacy of Graph Approximation



(a) Different δ (b) Different number of locations

Figure 11: Quality Loss of Graph Approximation

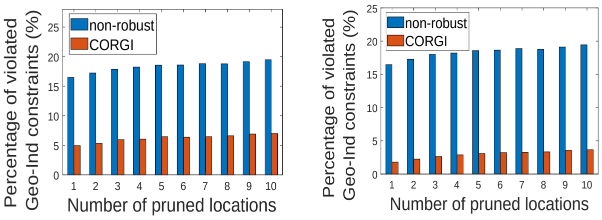


(a) D1 sample (b) D2 sample

Figure 12: Impact of privacy parameter (ϵ) and customization parameter (δ) on quality.

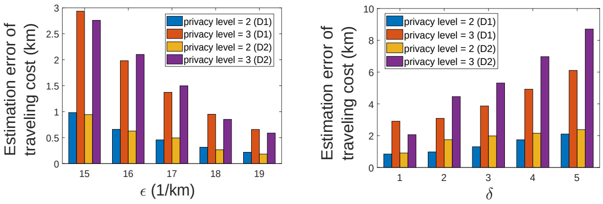
6.2.3 Impact of privacy parameters. In this experiment, we test the impact of privacy parameter ϵ and customization parameter δ on quality loss using prior probability distributions that are computed from two different regions/samples in the Gowalla dataset (D1, D2). When generating the matrix, we set $NR\_TARGET = 49$ (same as earlier) and used priors from D1 and D2 respectively. We compared our results against the baseline ("non-robust") approach which has $\delta = 0$ and therefore is not robust against pruning of any locations from the matrix, and depicts the results using D1 and D2 in Fig. 12(a)(b), respectively. In both Fig. 12(a)(b), the y-axis denotes the quality loss, and the x-axis denotes the ϵ value that ranges from 15/km to 20/km in increments of 1/km. As illustrated in both figures, (1) with increasing ϵ, the quality loss decreases, since higher ϵ implies weaker privacy and hence lower quality loss; (2) higher δ introduces higher quality loss, as higher privacy budget $\epsilon'_{i,j}$ is needed for each pair of real locations v_i and v_j (according to Equ. (19)) is needed to offset possible pruning of locations.

6.2.4 Impact of pruning locations. Users might not strictly follow the preferences that, only δ locations can be pruned (see Section 5.3). Therefore, in this experiment, we test the impact of the number of locations pruned on the quality loss, especially when this number is higher than δ. We create 10 experiment groups, wherein each group n ($n = 1, ..., 10$), we let a user randomly prune n locations from the leaf nodes of the location tree and run the experiment 500 times. We test both CORGI and the baseline and depict the results in Fig. 13(a)(b), where the number of locations is 49 and 70, respectively. In both figures, the x-axis denotes the number of locations pruned by a user, which is increased from 1 to 10, and the y-axis denotes the number of Geo-Ind constraint violations. As expected, the number of privacy violations in the non-robust matrix is much higher than that of the robust matrix. For example, pruning 14.28% locations only causes 3.07% Geo-Ind constraint violations in the matrix generated by CORGI, while it causes 18.58% Geo-Ind constraint violations in the non-robust matrix. We also observe that with higher δ, CORGI is more robust to the pruned locations as it preserves a higher privacy budget in the Geo-Ind constraints. The small number of privacy violations in some robust matrices is due to, 1) the number of pruned locations is greater than δ (the maximum number of locations expected to be removed) and 2) the robust matrix generation algorithm only converges to a relatively small threshold instead of 0 in consecutive iterations, indicating the output matrices might still have a small number of entries violating the preserved privacy budget.



(a) $\delta = 3$ (b) $\delta = 5$

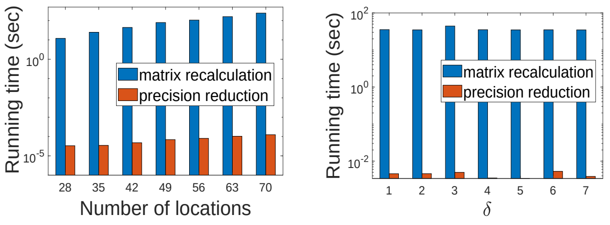
Figure 13: Impact of customization parameter (δ) on Geo-Ind violations.



(a) Quality loss with different ϵ (b) Quality loss with different δ

Figure 14: Impact of obfuscation range (privacy level) on quality loss.

6.2.5 Impact of privacy level on quality loss. In this experiment, we test the quality loss of CORGI given different privacy levels. The location tree has four levels, where level 3 includes the root node covering 343 locations, level 2, 1, and 0 includes 49 locations, 7 locations, and 1 location, respectively. Here, we compare two possible choices from users: ① privacy level = 3 (with precision level = 1), and ② privacy level = 2 (with precision level = 0). Fig. 14(a)(b) compares the quality loss of the two choices given different ϵ and δ values. Not surprisingly, the quality loss of both choices decreases with the increase of ϵ and increases with the increase of δ, which are consistent with the results in Fig. 11. Furthermore, the quality loss of privacy level 3 is higher than that of privacy level 2, since level 3 has a wider range of obfuscated locations to select for users (covering 343 leaf nodes) compared to level 2 (covering 343 leaf nodes), and hence leads to a higher distortion between estimation error of traveling cost.



(a) Running time with different number of locations

(b) Running time with different values of δ

Figure 15: Efficacy of precision reduction.

6.2.6 Computation time (precision reduction vs. matrix recalculation). Recall that in CORGI, the server first generates the obfuscation matrix for leaf level nodes. If a user selects to share location at a lower granularity, then instead of recalculating the matrix, CORGI generates the matrix via the precision reduction of the matrix at the bottom level. As such, in the last experiment, we test the computation time of precision reduction with the comparison of matrix recalculation. Fig. 15(a)(b) shows the running time of the two approaches given the different numbers of locations (from 28 to 70) and different δ (from 1 to 7). Both figures demonstrate that precision reduction can significantly reduce the computation time compared to matrix recalculation, e.g., on average, the computation time of precision reduction is only 0.000073% of that of the matrix recalculation.

7 RELATED WORK

Geo-I based obfuscation. The discussion of location privacy criteria can date back to almost two decades ago, when Gruteser and Grunwald [13] first introduced the notion of *location k-anonymity* on the basis of Sweeney's well-known concept of *k-anonymity* for data privacy [26]. Location k-anonymity was originally used to hide a user's identity in LBS [32]. This notion has been extended to obfuscate location by means of *l-diversity*, i.e., a user's location cannot be distinguished with other $l-1$ locations [29, 31]. However, l-diversity is hard to achieve in many applications as it assumes dummy locations are equally likely to be the real location from the attacker's view [2, 31].

In recent years, the privacy notion *Geo-Ind* [2] which was first by introduced by Andres et. al, and many obfuscation strategies based on it (e.g., [3, 6, 19, 24, 24, 25, 28, 31]) have been used for location obfuscation. As these strategies inevitably introduce errors to users' reported locations, leading to a quality loss in LBS, a key issue that has been discussed in those works is how to trade off QoS and privacy. Many existing works follow a global optimization framework: given the Geo-I constraints, an optimization model is formulated to minimize the quality loss caused by obfuscation [11, 12, 25, 28]. We now cover the related work closer to our work by categorizing them into tree-based approaches to obfuscation and policy-based approaches to customization.

Tree/hierarchy based approaches to location obfuscation. [1] uses a hierarchical grid to overcome the computational overhead of optimal mechanisms. They first construct a hierarchical grid with increasing granularity as one traverses down the index with the highest granularity at leaf nodes (similar to our approach). Second, they allocate the privacy budget (ϵ) appropriately to these different levels using sequential composition. In order to generate the obfuscated location, they start at the root node containing the real location of the user and go down the tree by recursively using the output of the obfuscation function at the prior level. The main difference between our approach and theirs is, they partition the privacy budget for each level in the grid, while ours, no matter from top to bottom or bottom to up (increase or decrease precision), uses the maximum privacy budget. In [27], the authors present a tree-based approach for differentially private online task assignments for crowdsourcing applications. They construct a Hierarchically well-Separated Tree (HST) based on a region that is published to both workers and task publishers who use it in order to obfuscate worker and task locations respectively. However, their approach relies on workers and task publishers using the same HST and obfuscation function in order to effectively perform task assignments and is not geared toward allowing users to customize the obfuscation functions. Other hierarchical-based approaches to spatial data such as [9, 23] focus on private release population statistics or histograms.

Policy based approach to privacy. Blowfish privacy proposed by [14] uses a policy graph to determine the set of neighbors that users want to mark as sensitive. A policy graph encodes the user's preferences about which pairs of values in the domain of the database should be indistinguishable for an adversary. Thus, it allows users to tradeoff privacy for utility by restricting the indistinguishability set. Blowfish works for statistical queries and not location queries. [7] extended blowfish privacy and applied it to location privacy where the nodes and edges in the policy graph represent possible locations of the user and the indistinguishability requirements respectively. Their goal is to ensure *ϵ-Geo-Ind* for any two connected nodes in the graph and to achieve this they apply DP-based noise to latitude and longitude independently. Their approach is best suited for category-based privacy i.e., indistinguishability among multiple locations of the same category (e.g., restaurants) as specifying pairwise indistinguishability between locations according to general user preferences is challenging. Our customization model allows users to specify their preferences which then get translated to the parameters in generating the obfuscation function reducing the overhead on the user in terms of specification. Furthermore, [7] does not allow users to choose the granularity at which their location is shared as the graph model doesn't capture the natural hierarchy of locations. [4] proposed an approach to recommend location privacy preferences based on place and time (similar to our customization policies) using local differential privacy. Their work is complementary to ours and could be used to help users in coming up with their user preferences.

8 CONCLUSIONS

We developed CORGI, a framework for generating customizable obfuscation functions with strong privacy guarantees via Geo-Indistinguishability. CORGI includes a location tree and a policy model to assist users in specifying their customization parameters. CORGI includes user and server-side interactions for efficiently generating a robust matrix. Experimental results show that CORGI effectively balances privacy, utility, and customization.

9 ACKNOWLEDGEMENT

This work was supported by an NSF grant under CNS-2136948. We thank the reviewers for their comments which helped to improve the paper.

REFERENCES

[1] Ritesh Ahuja, Gabriel Ghinita, and Cyrus Shahabi. 2019. A Utility-Preserving and Scalable Technique for Protecting Location Data with Geo-Indistinguishability. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*, Melanie Herschel, Helena Galhardas, Berthold Reinwald, Irini Fundulaki, Carsten Binnig, and Zoi Kaoudi (Eds.). OpenProceedings.org, 217–228. https://doi.org/10.5441/002/edbt.2019.20

[2] Miguel E. Andrés, Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. 2013. Geo-Indistinguishability: Differential Privacy for Location-Based Systems. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 901–914. https://doi.org/10.1145/2508859.2516735

[3] C. Ardagna, M. Cremonini, S. Vimercati, and P. Samarati. 2011. An Obfuscation-Based Approach for Protecting Location Privacy. *IEEE TDSC* 8 (03 2011), 13 – 27. https://doi.org/10.1109/TDSC.2009.25

[4] Maho Asada, Masatoshi Yoshikawa, and Yang Cao. 2019. "When and Where Do You Want to Hide?" - Recommendation of Location Privacy Preferences with Local Differential Privacy. In *Data and Applications Security and Privacy XXXIII - 33rd Annual IFIP WG 11.3 Conference, DBSec 2019, Charleston, SC, USA, July 15-17, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11559)*, Simon N. Foley (Ed.). Springer, 164–176. https://doi.org/10.1007/978-3-030-22479-0_9

[5] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data* (Atlantic City, New Jersey, USA) *(SIGMOD '90)*. Association for Computing Machinery, New York, NY, USA, 322–331. https://doi.org/10.1145/93597.98741

[6] Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. 2014. Optimal Geo-Indistinguishable Mechanisms for Location Privacy. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (Scottsdale, Arizona, USA) *(CCS '14)*. Association for Computing Machinery, New York, NY, USA, 251–262. https://doi.org/10.1145/2660267.2660345

[7] Yang Cao, Yonghui Xiao, Shun Takagi, Li Xiong, Masatoshi Yoshikawa, Yilin Shen, Jinfei Liu, Hongxia Jin, and Xiaofeng Xu. 2020. PGLP: Customizable and Rigorous Location Privacy Through Policy Graph. In *European Symposium on Research in Computer Security*. Springer International Publishing, 655–676.

[8] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. 2011. Friendship and Mobility: User Movement in Location-Based Social Networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Diego, California, USA) *(KDD '11)*. Association for Computing Machinery, New York, NY, USA, 1082–1090. https://doi.org/10.1145/2020408.2020579

[9] Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, Entong Shen, and Ting Yu. 2012. Differentially Private Spatial Decompositions. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, Anastasios Kementsietsidis and Marcos Antonio Vaz Salles (Eds.). IEEE Computer Society, 20–31. https://doi.org/10.1109/ICDE.2012.16

[10] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.* 9, 3-4 (2014), 211–407.

[11] Kassem Fawaz, Huan Feng, and Kang G. Shin. 2015. Anatomization and Protection of Mobile Apps' Location Privacy Threats. In *Proceedings of the 24th USENIX Conference on Security Symposium* (Washington, D.C.) *(SEC'15)*. USENIX Association, USA, 753–768.

[12] K. Fawaz and K. G. Shin. 2014. Location Privacy Protection for Smartphone Users. In *Proc. of ACM CCS* (Scottsdale, Arizona, USA). ACM, New York, NY, USA, 239–250. https://doi.org/10.1145/2660267.2660270

[13] Marco Gruteser and Dirk Grunwald. 2003. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services* (San Francisco, California) *(MobiSys '03)*. Association for Computing Machinery, New York, NY, USA, 31–42. https://doi.org/10.1145/1066116.1189037

[14] Xi He, Ashwin Machanavajjhala, and Bolin Ding. 2014. Blowfish privacy: tuning privacy-utility trade-offs using policies. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu (Eds.). ACM, 1447–1458. https://doi.org/10.1145/2588555.2588581

[15] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2011. What can we learn privately? *SIAM J. Comput.* 40, 3 (2011), 793–826.

[16] Daniel Kifer and Ashwin Machanavajjhala. 2014. Pufferfish: A framework for mathematical privacy definitions. *ACM Transactions on Database Systems (TODS)* 39, 1 (2014), 1–36.

[17] Jong Wook Kim, Kennedy Edemacu, Jong Seon Kim, Yon Dohn Chung, and Beakcheol Jang. 2021. A Survey Of differential privacy-based techniques and their applicability to location-Based services. *Computers & Security* 111 (2021), 102464. https://doi.org/10.1016/j.cose.2021.102464

[18] Primal Pappachan, Vishnu Sharma Hunsur Manjunath, Chenxi Qiu, Anna Squicciarini, and Hailey Onweller. 2023. CORGI: An interactive framework for Customizable and Robust Location Obfuscation. In *IEEE International Conference on Data Engineering (ICDE)*. IEEE.

[19] Chenxi Qiu, Anna Squicciarini, Zhuozhao Li, Ce Pang, and Li Yan. 2020. Time-Efficient Geo-Obfuscation to Protect Worker Location Privacy over Road Networks in Spatial Crowdsourcing. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event, Ireland) *(CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 1275–1284. https://doi.org/10.1145/3340531.3411863

[20] Chenxi Qiu, Anna Squicciarini, Zhuozhao Li, Ce Pang, and Li Yan. 2020. Time-Efficient Geo-Obfuscation to Protect Worker Location Privacy over Road Networks in Spatial Crowdsourcing. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event, Ireland) *(CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 1275–1284. https://doi.org/10.1145/3340531.3411863

[21] Chenxi Qiu, Anna Squicciarini, Ce Pang, Ning Wang, and Ben Wu. 2022. Location Privacy Protection in Vehicle-Based Spatial Crowdsourcing via Geo-Indistinguishability. *IEEE Transactions on Mobile Computing* 21, 7 (2022), 2436–2450. https://doi.org/10.1109/TMC.2020.3037911

[22] Chenxi Qiu, Li Yan, Anna Squicciarini, Juanjuan Zhao, Chengzhong Xu, and Primal Pappachan. 2022. TrafficAdaptor: An Adaptive Obfuscation Strategy for Vehicle Location Privacy against Traffic Flow Aware Attacks. In *Proceedings of the 30th International Conference on Advances in Geographic Information Systems* (Seattle, Washington) *(SIGSPATIAL '22)*. Association for Computing Machinery, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3557915.3560938

[23] Sina Shaham, Gabriel Ghinita, Ritesh Ahuja, John Krumm, and Cyrus Shahabi. 2021. HTF: Homogeneous Tree Framework for Differentially-Private Release of Location Data. In *Proceedings of the 29th International Conference on Advances in Geographic Information Systems* (Beijing, China) *(SIGSPATIAL '21)*. Association for Computing Machinery, New York, NY, USA, 184–194. https://doi.org/10.1145/3474717.3483943

[24] Reza Shokri, George Theodorakopoulos, Jean-Yves Le Boudec, and Jean-Pierre Hubaux. 2011. Quantifying Location Privacy. In *2011 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 247–262. https://doi.org/10.1109/SP.2011.18

[25] Reza Shokri, George Theodorakopoulos, Carmela Troncoso, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. 2012. Protecting Location Privacy: Optimal Strategy against Localization Attacks. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 617–627. https://doi.org/10.1145/2382196.2382261

[26] L. Sweeney. 2002. Achieving K-anonymity Privacy Protection Using Generalization and Suppression. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 10, 5 (2002), 571–588. https://doi.org/10.1142/S021848850200165X

[27] Qian Tao, Yongxin Tong, Zimu Zhou, Yexuan Shi, Lei Chen, and Ke Xu. 2020. Differentially Private Online Task Assignment in Spatial Crowdsourcing: A Tree-based Approach. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 517–528. https://doi.org/10.1109/ICDE48307.2020.00051

[28] Leye Wang, Dingqi Yang, Xiao Han, Tianben Wang, Daqing Zhang, and Xiaojuan Ma. 2017. Location Privacy-Preserving Task Allocation for Mobile Crowdsensing with Differential Geo-Obfuscation. In *Proceedings of the 26th International Conference on World Wide Web* (Perth, Australia) *(WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 627–636. https://doi.org/10.1145/3038912.3052696

[29] T. Wang and L. Liu. 2009. Privacy-aware Mobile Services over Road Networks. *VLDB Endow.* 2, 1 (Aug. 2009), 1042–1053. https://doi.org/10.14778/1687627.1687745

[30] Mengmeng Yang, Lingjuan Lyu, Jun Zhao, Tianqing Zhu, and Kwok-Yan Lam. 2020. Local Differential Privacy and Its Applications: A Comprehensive Survey. *CoRR* abs/2008.03686 (2020). arXiv:2008.03686 https://arxiv.org/abs/2008.03686

[31] Lei Yu, Ling Liu, and Calton Pu. 2017. Dynamic Differential Location Privacy with Personalized Error Bounds. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society. https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/dynamic-differential-location-privacy-personalized-error-bounds/

[32] Lijuan Zheng, Huanhuan Yue, Zhaoxuan Li, Xiao Pan, Mei Wu, and Fan Yang. 2018. k-Anonymity Location Privacy Algorithm Based on Clustering. *IEEE Access* 6 (2018), 28328–28338. https://doi.org/10.1109/ACCESS.2017.2780111