Knowledge and Information Systems (2020) 62:2613–2651
https://doi.org/10.1007/s10115-019-01432-4

REGULAR PAPER

Location histogram privacy by Sensitive Location Hiding and Target Histogram Avoidance/Resemblance

Grigorios Loukides[1] · George Theodorakopoulos[2]

Received: 15 August 2018 / Revised: 4 December 2019 / Accepted: 8 December 2019 /
Published online: 31 December 2019

Abstract
A location histogram is comprised of the number of times a user has visited locations as they move in an area of interest, and it is often obtained from the user in the context of applications such as recommendation and advertising. However, a location histogram that leaves a user's computer or device may threaten privacy when it contains visits to locations that the user does not want to disclose (*sensitive locations*), or when it can be used to profile the user in a way that leads to price discrimination and unsolicited advertising (e.g., as "wealthy" or "minority member"). Our work introduces two privacy notions to protect a location histogram from these threats: *Sensitive Location Hiding*, which aims at concealing all visits to sensitive locations, and *Target Avoidance/Resemblance*, which aims at concealing the similarity/dissimilarity of the user's histogram to a *target histogram* that corresponds to an undesired/desired profile. We formulate an optimization problem around each notion: Sensitive Location Hiding (*SLH*), which seeks to construct a histogram that is as similar as possible to the user's histogram but associates all visits with nonsensitive locations, and Target Avoidance/Resemblance (*TA*/*TR*), which seeks to construct a histogram that is as dissimilar/similar as possible to a given target histogram but remains useful for getting a good response from the application that analyzes the histogram. We develop an optimal algorithm for each notion, which operates on a notion-specific search space graph and finds a shortest or longest path in the graph that corresponds to a solution histogram. In addition, we develop a greedy heuristic for the *TA*/*TR* problem, which operates directly on a user's histogram. Our experiments demonstrate that all algorithms are effective at preserving the distribution of locations in a histogram and the quality of location recommendation. They also demonstrate that the heuristic produces near-optimal solutions while being orders of magnitude faster than the optimal algorithm for *TA*/*TR*.

Keywords Location privacy · Histogram privacy · Location-based services · Dynamic programming

✉ George Theodorakopoulos
TheodorakopoulosG@cardiff.ac.uk

Grigorios Loukides
grigorios.loukides@kcl.ac.uk

1 Department of Informatics, King's College London, London, UK

2 School of Computer Science and Informatics, Cardiff University, Cardiff, UK

1 Introduction

A location histogram is a statistical summary of a user's whereabouts, comprised of the number of times a user has visited each location in an area of interest. Location histograms are often obtained from users, in the context of applications including recommendation [28,29,65], advertising [14,21], and location pattern discovery [64]. For example, a recommender application typically employs a set of location histograms each corresponding to a different user (i.e., a user-location matrix) as a training set, and it aims at recommending locations that a user may be interested in visiting based on the user's histogram [65]. Location histograms are also often visualized or analyzed directly [68].

However, a location histogram that leaves a user's computer or device may pose a threat to the user's privacy. This happens when the histogram contains visits to *sensitive* locations that the user does not want to disclose, because they are associated with confidential information (e.g., a temple is associated with a religion, and the headquarters of a political organization with certain political beliefs), or when the histogram can be used to profile the user (e.g., as "wealthy" or "minority member") leading to price discrimination [37,38] and unsolicited advertising [5]. For example, if the histogram reveals that a user frequently visits expensive restaurants, a targeted-advertisement application may display to the user advertisements about products and services that are priced higher than normal [37,38].

In this work, we introduce two novel notions of histogram privacy, Sensitive Location Hiding and Target Avoidance/Resemblance, for protecting against the disclosure of sensitive locations and user profiling, respectively. Sensitive Location Hiding aims at concealing all visits to user-specified sensitive locations, by producing a *sanitized* histogram, in which the frequencies associated with the sensitive locations are equal to zero. This protects a user from an adversary who receives the sanitized histogram, knows the set of locations considered to be sensitive, and tries to infer which of these sensitive locations were visited by the user. By enforcing the notion of Sensitive Location Hiding, users are able to disseminate their location histogram in order to benefit from location-based services, such as location recommendation, while being protected from the inference of their sensitive locations and the aforementioned consequences such inference may have.

Target Avoidance aims at concealing the fact that the user's histogram is similar to an undesirable histogram that, if disseminated, would lead to undesired user profiling. For example, a user may wish to make their histogram dissimilar to a target histogram of a typical wealthy person, containing frequent visits to expensive restaurants, to avoid price discrimination [37]. As another example, a user's location histogram may allow the inference of the user's political affiliation, religious beliefs, and sexual orientation, which may lead to emotional distress, harassment, or even persecution. Thus, a user would wish to avoid disseminating a histogram that is similar to histograms that can lead to such undesirable inferences. This protects from adversaries who use the sanitized histogram and the target histogram of a person with an undesirable profile, to infer that the user's histogram resembles the latter histogram.

Target Resemblance is a variant of Target Avoidance, in which the user expressly *wishes* to make their histogram similar to the target histogram representing a desirable profile. For example, the desirable target histogram for a tourist could be that of a local resident in order to avoid discriminatory practices toward tourists (e.g., price discrimination). As another example, consider a company that engages in secret discriminatory hiring practices by preferentially hiring members of a particular demographic group. There are cases where companies have been shown to discriminate based on sexual orientation when hiring [57]. In these cases, a person who wishes to be hired will want to make their histogram resemble that of an heterosexual person, so as to avoid discriminatory treatment. The target histogram may be specified by the users themselves, or selected with the help of domain experts (see Sect. 3.3). Enforcing Target Resemblance protects from adversaries who use the sanitized histogram and the target histogram of a person with a desirable profile, to infer that the user's histogram does not resemble the latter histogram.

Comparing Target Avoidance and Target Resemblance, we see that in both cases the adversary aims to infer whether or not the sanitized histogram resembles a given target histogram. The difference is that, in Target Avoidance, the user wants the adversary to conclude that there is no resemblance, whereas in Target Resemblance the user wants the opposite.

Our privacy notions can be achieved by histogram sanitization, i.e., by changing the frequencies of location visits in the histogram. However, sanitization incurs a quality (utility) loss, which must be controlled to ensure that the user obtains a good response from the application which uses their sanitized histogram. To achieve this balance between privacy and quality, we define an optimization problem around each privacy notion: the Sensitive Location Hiding (*SLH*) problem, which seeks to construct a sanitized histogram with minimum quality loss, and the Target Avoidance/Resemblance (*TA*/*TR*) problem, which seeks to avoid/resemble the target to a level at least equal to a user-provided privacy parameter, while ensuring that the quality loss does not exceed a user-provided quality parameter. If it is impossible to satisfy both the privacy and the quality requirements, then the problem has no solution.

Neither notion can be achieved by existing methods for histogram sanitization. The aim of existing methods is to either (I) prevent the inference of the exact frequencies of the histogram (i.e., the number of visits to one or more locations) [2,16,23,26,45,61,68], or (II) make a user's histogram indistinguishable from a set of histograms belonging to other users [18,20,60]. Their aim is neither to hide sensitive locations, nor to avoid/resemble a target histogram. The privacy notions we introduce in the paper are important to achieve in real applications, as we discuss in Examples 2.1 and 2.2 in Sect. 2.

Therefore, we develop new methods for achieving the *SLH* and the *TA*/*TR* notions: (I) an optimal algorithm for *SLH*, called *LHO* (Location Hiding Optimal), (II) an optimal algorithm for *TR*, called *RO* (Resemblance Optimal), and (III) a greedy heuristic for *TR*, called *RH* (Resemblance Heuristic). Because *TA* and *TR* are similar, we focus on *TR* and discuss *TA* briefly.

Our methods are both effective and efficient, as demonstrated by experiments using two real datasets derived from the Foursquare location-based social network [62], which together contain approximately 3400 histograms. In terms of effectiveness, all algorithms achieve the corresponding notions, or announce that it is impossible to achieve them, and they are additionally able to preserve: (I) the distribution of locations in a histogram, which is useful in applications such as aggregate query answering and classification [32,68], and (II) the quality of location recommendation based on collaborative filtering [35]. In addition, the heuristic produces near-optimal solutions (up to 1.5% worse than the optimal), with respect to preserving distribution similarity. In terms of efficiency, all algorithms scale well with the histogram parameters, requiring from less than 1 second (the *LHO* algorithm) to 5 minutes (the *RO* algorithm). In addition, the *RH* heuristic is more efficient than the optimal algorithm by at least two orders of magnitude.

We note that our notions are framed in the context of location histograms but can be applied to any histogram. For example, they could be applied to a histogram comprised of webpage visits. The resultant sanitized histogram would then conceal visits to webpages that a user does not want to disclose, or it would resemble/avoid a target histogram for protecting the user from targeted advertising based on their webpage visits.

Organization We provide an overview of and motivation for our approach in Sect. 2; we introduce formal notation, and we formalize the privacy notions, the adversary models, and the optimization problems we solve in Sect. 3; we describe our algorithms and our heuristics in Sect. 4; we evaluate our approach in Sect. 5; we discuss related work in Sect. 6; and we conclude the paper in Sect. 7.

2 Overview and motivation of our approach

This section provides examples to motivate the need for Sensitive Location Hiding and Target Resemblance and also provides a high-level overview of the optimization problems and methods for solving them.

2.1 Sensitive Location Hiding

Given a set of sensitive locations, a histogram satisfies the notion of Sensitive Location Hiding when the frequency of each of its sensitive locations is zero. Clearly, one simple strategy to achieve this notion is by setting the frequency of each sensitive location of a given histogram to zero. However, this strategy may have a substantial negative impact on the quality (utility) of the histogram in location histogram applications. This is because it reduces the *size* (sum of frequencies) of the histogram. A size reduction should be avoided because some important statistics depend on the size of the histogram. An example of such statistics is the fraction of all users' visits to a particular location in a city (i.e., the ratio between the sum of the frequency of the location over all users' histograms and the sum of the sizes of these histograms), which is a simple indicator of the popularity of the location. Another example is the average number of visits to a location (i.e., the ratio between the size of the user's histogram and the number of locations in the histogram), which is used in location recommendation [7,35].

A different strategy that achieves the Sensitive Location Hiding notion, while preserving the size of the histogram, is to redistribute the frequency counts of the sensitive locations to nonsensitive ones. However, the redistribution needs to be performed in a way that preserves the quality (utility) of the histogram in location histogram applications. The impact of each possible redistribution on quality must be quantified, and the selected redistribution strategy must be the one with the lower impact. We quantify the impact of a redistribution strategy with a quality distance function, similarly to most works on histogram sanitization [2,26,61,68]. This function offers generality, because different functions can be chosen for different applications.

The above discussion motivates the formulation of the *Sensitive Location Hiding* (*SLH*) optimization problem: given a histogram H, a set of sensitive locations, and a quality distance function, produce a sanitized histogram H' such that the frequency of each sensitive location in H' is 0, H' is as similar as possible to H, and H' has the same size as H. Similarity is measured with the quality distance function.

In Sect. 3.2, we give a formal definition of the *SLH* optimization problem, discuss the adversary model it provides protection against, and show that the problem is *weakly* NP-hard [41]. In addition, we discuss a variation of the problem which relaxes the size requirement and can be easily dealt with by our algorithms.

To illustrate the *SLH* notion and the *SLH* problem, we now provide Example 2.1, which is inspired from approaches on privacy-preserving recommendation [48,58]. However, the *SLH* notion and problem are not tied to recommendation and cannot be handled with existing approaches.

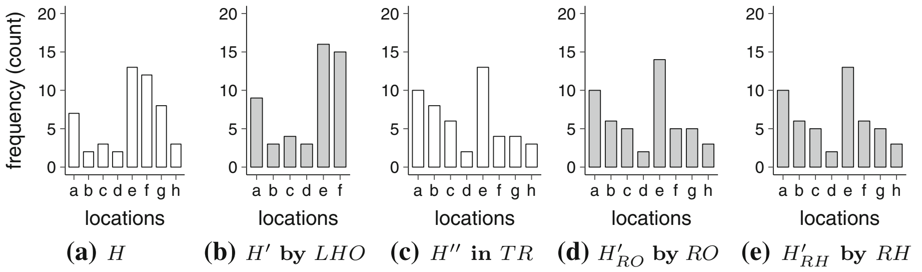


Fig. 1 **a** Location histogram. **b** Sanitized histogram produced by the *LHO* algorithm when g and h are sensitive locations and the quality distance function is Jensen–Shannon divergence [30] (only positive counts are illustrated). **c** Target histogram used in the *TR* problem. **d** Sanitized histogram produced by the *RO* algorithm when applied with the target histogram in **c**. **e** Sanitized histogram produced by the *RH* heuristic when applied with the target histogram in (**c**)

Example 2.1 (Illustration of the *SLH* notion and *SLH* problem) An application provides location recommendations to users by analyzing their location profiles. To obtain a recommended location, a user must send 50 location visits to the application in the form of a location histogram. To compute the recommended location, the application uses common mining tasks, such as discovering frequent location patterns in the user's histogram and finding similar histograms to it [67]. The location histogram H of a user Alice is shown in Fig. 1a. The histogram contains the number of times Alice visited each of the locations a to h. Alice is not willing to provide H to the application, because the last two locations in H, g and h, are sensitive, but she still wishes to receive a "good" recommended location by the application. Therefore, Alice solves the *SLH* problem and obtains the sanitized histogram H' shown in Fig. 1b. The sanitized histogram preserves privacy, because it does not contain the sensitive locations. It can be sent to the application to receive a fairly accurate recommendation, because it contains 50 visits to nonsensitive locations (the visits to sensitive locations are zero and not shown) and is as "similar" as possible to H, to the extent permitted by the privacy requirement. □

To optimally solve the *SLH* problem, the *LHO* algorithm finds the exact number of sensitive location visits that need to be redistributed into each *nonsensitive bin* (bin corresponding to a nonsensitive location), so that all sensitive location visits are redistributed and quality is optimally preserved, with respect to the quality distance function. That is, the algorithm determines the frequency of each nonsensitive location of the sanitized histogram H', so that H' has the same size with the given histogram H and is as similar as possible to it, with respect to the quality distance function. However, it is computationally prohibitive to directly compute the quality of each possible redistribution of the sensitive location visits into the nonsensitive bins and then select the optimal solution. This follows from the fact that there are $O\left(\binom{K+m-1}{m-1}\right)$ ways to redistribute K sensitive location visits into m nonsensitive bins (each way corresponds to a *weak composition* of K [6]). Therefore, *LHO* solves the problem by modeling it as a shortest path problem between two specific nodes, s and t, of a directed acyclic graph (DAG) (see Fig. 2). The node s is labeled $(0, 0)$, and each other node is labeled (i, j), where $i \in [1, m]$ corresponds to a nonsensitive location L_i and $j \in [0, K]$ corresponds to the number of sensitive location visits that will be redistributed into the nonsensitive bins $1, \ldots, i$ of the sanitized histogram H'. For example, the label (m, K) of the node t denotes the redistribution of all K sensitive location visits to all m nonsensitive bins of H'. We may refer to a node using its label. The graph contains an edge from each

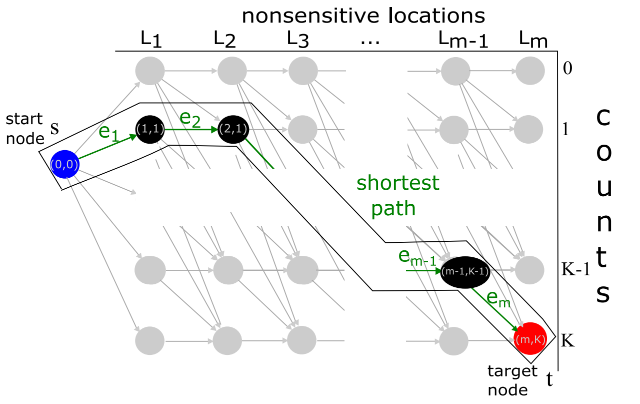


Fig. 2 Graph constructed by the Location Hiding Optimal (*LHO*) algorithm. The shortest path from s to t corresponds to the optimal solution of the *SLH* problem. Each node (i, j), $i \in [1, m]$, $j \in [0, K]$, in the path denotes the redistribution of j sensitive location visits into the nonsensitive bins $1, \ldots, i$. The path corresponds to the optimal way of redistributing all K sensitive location visits into all m nonsensitive bins. The weight of the edge $((i, j), (i + 1, j + k))$ denotes the impact on quality caused by redistributing k sensitive location visits into the nonsensitive bin $i + 1$, and the sum of the edge weights of this path $e_1 + \cdots + e_m$ quantifies the quality distance between the optimal solution and H

node (i, j) to each node $(i + 1, j + k)$ with $k \in [0, K - j]$, where k denotes the number of sensitive location visits that are redistributed into the nonsensitive bin $i + 1$. For example, the edge $((i, j), (i + 1, j + k)) = ((1, 1), (2, 1))$ denotes that $k = 0$ visits are redistributed into the nonsensitive bin $i + 1 = 2$. Each edge $((i, j), (i + 1, j + k))$ has a weight that quantifies the impact on quality caused by the redistribution of k sensitive location visits into the nonsensitive bin $i + 1$. Every path from s to t corresponds to a feasible solution of the *SLH* problem. This is because the nodes in the path uniquely determine how all K sensitive location visits will be redistributed into all m nonsensitive bins of H' (see property (I) in Sect. 4.1). In addition, the length (sum of edge weights) of the path is equal to the quality distance between the corresponding solution H' and H (see property (II) in Sect. 4.1). Thus, the shortest path from s to t corresponds to a histogram H' that is as similar as possible to H, and therefore, it is the optimal solution of the *SLH* optimization problem. For example, applying the *LHO* algorithm to the histogram of Fig. 1a, when the locations g and h are sensitive and the quality distance function is Jensen–Shannon divergence (see Sect. 3.1.1), produces the sanitized histogram in Fig. 1b. Note that the visits to g and h are redistributed into all nonsensitive bins, so that the sanitized histogram is as similar as possible to the histogram of Fig. 1a. A formal description of the *LHO* algorithm, as well as the analysis of the algorithm is provided in Sect. 4.1.

2.2 Target Resemblance

Given a target histogram, a histogram satisfies the notion of Target Resemblance when it is similar enough to the target. A privacy distance function quantifies similarity, and a privacy parameter quantifies the threshold for determining whether the two histograms are similar enough.

Clearly, any histogram can be easily modified to be arbitrarily similar to a given target histogram, by simply redistributing all its frequency counts so that they are exactly equal to the counts in the target histogram. However, as in the case of *SLH*, a simplistic redistribution can deteriorate quality unacceptably. The modification to the histogram must balance between resemblance to the target histogram and similarity to the original histogram. A quality distance function quantifies the quality loss caused by the modification, and a quality parameter quantifies the threshold for determining whether the loss is acceptable or not.

The above discussion motivates the formulation of the *Target Resemblance* (*TR*) optimization problem: given a histogram H, a target histogram H'', a quality distance function and a quality parameter ϵ, a privacy distance function and a privacy parameter c, produce a sanitized histogram H' such that its quality distance from H is at most ϵ, its privacy distance from H'' is minimized, and its size is the same as H. If the resulting privacy distance of H' from H'' is larger than c, then there is no solution.

In Sect. 3.3, we give a formal definition of the *TR* problem, discuss the adversary model it provides protection against, and we show that it is *weakly* NP-hard. In addition, we discuss a variation that relaxes the size requirement and can be easily dealt with by our algorithms. To illustrate the *TR* privacy notion and optimization problem, we provide Example 2.2.

Example 2.2 (Illustration of the TR notion and problem, continuing from Example 2.1) Fig. 1a shows the location histogram H of a user, Bob, who wants to use the location recommendation application. Bob is not willing to provide H to the application, because he is concerned about price discrimination, as a result of frequent visits to locations f ("airport") and g ("5-star hotel"). To achieve his purpose, Bob can solve the Target Resemblance (*TR*) problem to generate a histogram that resembles the target histogram H'' in Fig. 1c. H'' reflects a budget-conscious person, because in H'' the frequencies of locations a ("train station"), b ("2-star hotel"), and c ("3-star hotel") are relatively high, whereas the frequencies of f and g are relatively low. Hence, H'' is likely to attract lower-priced recommendations than H would, and it is definitely more likely to prevent price discrimination [37,38]. The resemblance to H'' is satisfied by generating a *sanitized* histogram H'_{RO} (*RO* for "Resemblance Optimal") that minimizes a *privacy distance function* between the sanitized histogram and H''. In parallel, Bob still wishes to receive a "good" recommended location by the application. This quality requirement is satisfied by limiting the dissimilarity between H and the sanitized histogram H'_{RO} to a maximum of $\epsilon = 0.05$, as measured by a *quality distance function*, so that the sanitization preserves the similarity between H and other users' histograms, which helps compute a "good" recommended location [35]. After solving the *TR* problem, Bob obtains the sanitized histogram H'_{RO} in Fig. 1d, which is almost identical to the target H''. □

To optimally solve the *TR* problem, the Resemblance Optimal (*RO*) algorithm finds the exact number of location visits that need to be added into, or removed from, each bin of a histogram H, so that the resultant sanitized histogram H' is as similar as possible to the target histogram H'', and no more dissimilar from H than what is allowed by the quality threshold ϵ. Again, the large number of potential solutions, given by $O\left(\binom{N+n-1}{n-1}\right)$, where N is the size of H and n is its length, prohibits directly computing the quality of each possible solution and selecting the optimal solution. Therefore, *RO* solves the problem by modeling it as a constrained shortest path problem in a DAG (see Fig. 3). The graph contains a path $(u_0^0, u_1^{N_1}, \ldots, u_n^{N_n})$ for each allocation of $N = N_n$ counts to the n bins of the histogram (i.e., each allocation corresponds to a possible solution to the Target Resemblance problem, ignoring the quality constraint), where a node $u_i^{N_i}$ corresponds to allocating N_i counts to bins 1 up to and including i. The length of a path is equal to the dissimilarity of the corresponding allocation to the target histogram H'', whereas the cost of the path is equal to the quality loss

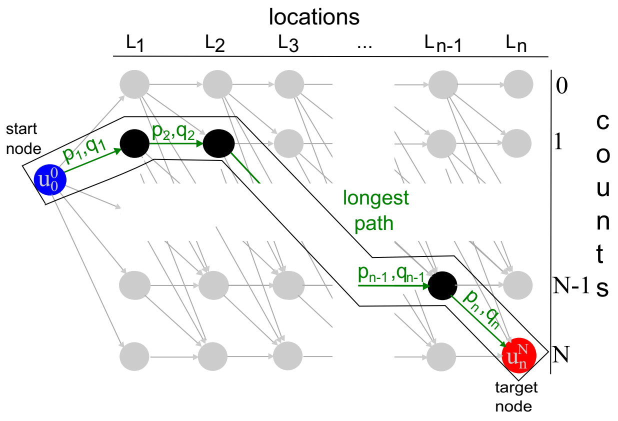


Fig. 3 Graph constructed by the Resemblance Optimal (*RO*) algorithm. The shortest path from u_0^0 to u_n^N with cost at most ϵ corresponds to the optimal solution of the *TR* problem. The nodes of this path correspond to the optimal way of allocating counts to all bins. The edge weights of this path are (privacy, quality loss) pairs $(p_1, q_1), \ldots, (p_n, q_n)$. The two weights of an edge $u_i^{N_i}$ to $u_{i+1}^{N_i+k}$ are the privacy and quality effects of allocating exactly k counts to bin $i + 1$ of the solution histogram. The sum $\sum_{i\in[1,n]} p_i$ quantifies the dissimilarity between the optimal solution and the target histogram (smaller is better) and the sum $\sum_{i\in[1,n]} q_i$ quantifies the total quality loss, which should be at most ϵ

as compared to the user's histogram H. The algorithm finds the shortest path among those whose cost does not exceed the quality threshold ϵ. As the graph is a DAG, to find the optimal solution it suffices to explore it in breadth-first search order. First, we compute constrained shortest paths to all nodes that correspond to bin 1: $u_1^{N_1}$, $N_1 = 0, \ldots, N$; then, we extend these paths to all nodes that correspond to bin 2: $u_2^{N_2}$, $N_2 = 0, \ldots, N$ and we prune them if they violate the quality constraint; we continue all the way to $u_{n-1}^{N_{n-1}}$, $N_{n-1} = 0, \ldots, N$ and finally to the node $u_n^{N_n}$, $N_n = N$. The shortest path to that final node corresponds to the optimal valid allocation of N counts to bins $1, \ldots, n$.

When solution optimality is not necessary, the *TR* problem can be solved more efficiently by the *RH* heuristic. *RH* differs from the *RO* algorithm in that it restricts the set of bins in the histogram H whose number of location visits can increase or decrease. Specifically, it works in a greedy fashion, iteratively "moving" frequency counts from *source bins* to *destination bins*. The source bins have higher frequency in H than in the target histogram H'', whereas the destination bins have lower frequency in H than in H''. Thus, moving counts from source to destination bins makes the sanitized histogram more and more similar to the target histogram, but it incurs a quality loss due to changes in frequency counts. Therefore, to control the loss of quality, moves are performed for as long as the quality distance of the resultant sanitized histogram from H does not exceed the quality threshold. Example 2.3 below illustrates the *RO* algorithm and the *RH* heuristic.

Example 2.3 (Illustration of the *RO* algorithm and *RH* heuristic, continuing from Example 2.2) Bob applies *RO* to his histogram H in Fig. 1a, using the target histogram H'' in Fig. 1c, the quality threshold $\epsilon = 0.05$, and the Jensen–Shannon divergence to measure dissimilarity from H'' and from H. The algorithm produces the sanitized histogram H'_{RO} in Fig. 1d, which is as similar to H'' as allowed by the specified threshold. Similarly, Bob applies *RH* and obtains the sanitized histogram H'_{RH} in Fig. 1e. Comparing H'_{RO} and H'_{RH} to H'', we observe that H'_{RO} is very similar to H'', while H'_{RH} is slightly less similar (e.g.,

Table 1 Acronyms and notation

Acronym	Meaning
SLH	Sensitive Location Hiding (input: H, L', $d_q()$, output: H')
LHO	Location Hiding Optimal (optimal algorithm for SLH)
TR	Target Resemblance (input: H, H'', $d_p()$, $d_q()$, ϵ, output: H')
RO	Resemblance Optimal (optimal algorithm for TR)
RH	Resemblance Heuristic (heuristic for TR)
TA	Target Avoidance (input: H, H'', $d_p()$, $d_q()$, ϵ, output: H')
JS	Jensen–Shannon divergence (distance function used in the evaluation)

Notation	Meaning
$L = \{L_1, \dots, L_{\|L\|}\}$	Set of semantic locations that a user can visit
$L' \subseteq L$	Set of sensitive locations (for SLH)
$\mathcal{H}_{n,N}$	Set of all histograms of length n, $n \leq \|L\|$, and size N
$H = (f(L_1), \dots, f(L_n))$	Histogram of nonnegative frequencies (counts)
	of visits to locations $L_1, \dots, L_n \in L, n \leq \|L\|$
H'	Sanitized histogram
H''	Target histogram (for TA and TR)
ϵ	Maximum quality loss threshold (for TA and TR)
$d_p(H', H'')$	Privacy distance function (for TA and TR)
$d_q(H, H')$	Quality distance function (measures quality loss)

the frequencies of f and g are equal in H'' and H'_{RO}, while they are not equal in H'' and H'_{RH}). However, H'_{RH} is still useful for getting a good recommendation from the application, because the quality loss (dissimilarity to H) does not exceed ϵ. □

3 Background, problem definitions, and adversary models

In this section, we define some preliminary concepts, and then, we formally define the *SLH*, *TA*, and *TR* optimization problems. A summary of the most important notation we introduce is in Table 1.

3.1 Preliminaries

We consider an area of interest, modeled as a finite set of semantic locations $L = \{L_1, \dots, L_{|L|}\}$ of cardinality $|L|$, where a location L_i, $i \in [1, |L|]$, is, e.g., "Italian Restaurant," "Cinema," or "Museum."

We also consider a user who moves in this area. The user's *histogram* is a vector of integer frequencies $H = (f(L_1), \dots, f(L_n))$, where $n \leq |L|$ is the *length* of the histogram. Each location L_i, $i \in [1, n]$, has a frequency $f(L_i) > 0$, when L_i was visited by the user, or $f(L_i) = 0$ otherwise. We may refer to frequencies as counts.

We use $H[i]$ to refer to the i-th element, or bin, of H, and N, or *size*, to refer to the L_1-norm $|H|_1 = \sum_{i \in [1,n]} H[i]$ of H. We use $\mathcal{H}_{n,N}$ to denote the set of all histograms of length n and size N.

Having compiled H, the user wishes to submit it to a location-based application. Before submitting it, the user transforms it into a *sanitized* histogram H' (in a way to be made concrete in Problems 3.1 and 3.2 below) and then submits H' to the application. Next, the application returns a response to the user. Depending on the sanitization required, H' may contain zero-frequency counts for some locations, or it may contain nonzero frequency counts for locations that the user never visited. If the user wishes, we can easily guarantee that H' will not contain nonzero frequency counts for locations that the user never visited, by assigning an infinite cost $d_q(H[i], H'[i])$ for each such location L_i.

3.1.1 Quality loss

Since the user submits H', which is in general different from H, there will be a negative impact on the quality of the application response. The resulting loss in quality is measured by a *quality distance* function $d_q(H, H')$. For every pair H, H', we require that $d_q(H, H') \geq 0$, and that $H = H'$ implies $d_q(H, H') = 0$. In addition, we require d_q to decompose as a sum over bins, i.e., there must be a function q such that $d_q(H, H') = \sum_{i \in [1,n]} q(H[i], H'[i])$. Most distances used in data mining applications in which distances between histograms/vectors must be preserved (e.g., Jensen–Shannon divergence (*JS*-divergence) [30], Jeffrey's divergence [45], L_2-distance (Euclidean distance) and squared Euclidean distance [66], variational distance [30], Pearson χ^2 distance [30], and Neyman χ^2 distance [30]) decompose as a sum over bins.

We use *JS*-divergence as the objective function d_q in our experiments (see Sect. 5). *JS*-divergence is a standard measure for quantifying distances between probability distributions, which is often used in histogram/vector classification [36] and clustering [40]. Given two histograms H_1, H_2, the JS-divergence between them is defined as

$$JS(H_1, H_2) = \frac{1}{2 \cdot N} \cdot \sum_{i \in [1,n]} \left(H_1[i] \cdot \log_2 \left(\frac{2 \cdot H_1[i]}{H_1[i] + H_2[i]} \right) + H_2[i] \cdot \log_2 \left(\frac{2 \cdot H_2[i]}{H_1[i] + H_2[i]} \right) \right).$$

with the convention $0 \cdot \log_2(0) = 0$. *JS*-divergence is bounded in $[0, 1]$ [30], and $JS(H_1, H_2) = 0$ implies no quality loss. As explained in [27], *JS*-divergence can also be easily extended to capture semantic similarity requirements (e.g., An Italian Restaurant is more similar to a French Restaurant than to an American Cinema), when this is needed in applications. The extended measure, called *smoothed JS*-divergence, requires preprocessing the histogram by kernel smoothing and then applying *JS*-divergence to the preprocessed histogram. Incorporating smoothed *JS*-divergence in our methods is straightforward and left for future work.

3.2 The sensitive Location Hiding problem: adversary model and formal definition

As discussed in the Introduction and in Sect. 2, the Sensitive Location Hiding (*SLH*) privacy notion aims to conceal all visits to sensitive locations. We formulate the adversary model and the desired privacy property for the *SLH* notion as follows.

The adversary knows: (I) the sanitized histogram H' that the user submits, (II) the set of all possible sensitive locations L', and (III) the fact that, if H' is fake, then it must have been produced by the *LHO* algorithm in our paper. The adversary has no other background knowledge. The adversary succeeds if, based on their knowledge, they manage to determine whether or not the user visited one or more of the sensitive locations in L'.

The desired privacy property is the negation of the adversary's success criterion. That is, the adversary must not be able to infer, from the sanitized histogram, that the user has visited any of the sensitive locations.

We formally define the corresponding optimization problem as follows:

Problem 3.1 (Sensitive Location Hiding (*SLH*)) Given a histogram $H \in \mathcal{H}_{n,N}$, a subset $L' \subseteq L$ of *sensitive* locations, and a quality distance function $d_q()$, construct a sanitized histogram $H' \in \mathcal{H}_{n,N}$ that

$$\begin{aligned} \underset{H' \in \mathcal{H}_{n,N}}{\text{minimizes}} \quad & d_q(H, H') \\ \text{subject to} \quad & H'[i] = 0 \text{ for each location } L_i \in L'. \end{aligned}$$

Intuitively, the *SLH* problem requires constructing a sanitized histogram by redistributing the counts of the sensitive locations of H into bins that correspond to nonsensitive locations, in the best possible way according to d_q. The sensitive locations are specified by the user based on their preferences.

In the *SLH* problem formulation, we follow the user-centric (or personalized) approach to privacy that is employed in [1,3,13,50]. This approach requires the users to specify their own privacy preferences, so that these preferences are best reflected in the produced solutions. However, not all users may possess knowledge allowing them to identify certain locations in their histograms as sensitive. Yet, such users often know that a class of locations are sensitive, or they do not want to be associated with a class of locations [31,59]. For instance, several users may not want to be associated with visits to any type of clinic or adult entertainment location. In this case, users may employ a taxonomy[1] to identify classes of sensitive locations, which requires less detailed knowledge. This method is inspired by [31,59] and simply requires a user to select one or more nodes in the taxonomy. If a node u that is not a leaf is selected, then all locations corresponding to leaves in the subtree rooted at u will be considered as sensitive. If the selected node u is a leaf, then its corresponding location will be considered as sensitive. Such taxonomies already exist for location-based data, and they can also be automatically constructed based on machine learning techniques [55]. For example, in the Foursquare taxonomy (see Sect. 5), there is an aggregate category (internal node) "Medical center" which contains more specific categories (leaves) "Hospital," "Rehab center," etc.

Theorem 3.1 *The SLH problem is weakly NP-hard.*

Proof See Appendix (Sect. A.1). □

Clearly, the *SLH* problem seeks to produce a sanitized histogram H' with the same size as H. As discussed in the Introduction, this allows preserving statistics that depend on the size of the histogram, which are important in location-based applications, such as location recommendation. However, it is also possible to require the sanitized histogram H' to have a given size instead (e.g., when an application requires a histogram to have a certain number of location counts, or in pathological cases where redistribution leads to undesirable/implausible histograms). This leads to a variation of the *SLH* problem, referred to as SLH_r, which requires redistributing $r \geq 0$ counts of sensitive locations into the bins corresponding to nonsensitive locations. Note the following choices for r in SLH_r: (I) for $r = 0$, the SLH_r problem requires constructing a sanitized histogram where each sensitive location has count 0 and

[1] A taxonomy (also known as hierarchy) is a tree structure, in which the root corresponds to the most general location "any," there is a leaf node for each sensitive location, and the internal nodes of the taxonomy correspond to aggregate/coarse sensitive locations.

each nonsensitive location has a count equal to that of its count in H. Such a histogram is trivial to produce, by simply replacing the count of each sensitive location with 0. (II) For $r = \sum_{L_i \in L'} f(L_i)$ (i.e., equal to the total count of sensitive locations), SLH_r becomes equivalent to the *SLH* problem. (III) For $r > \sum_{L_i \in L'} f(L_i)$, the SLH_r problem requires constructing a sanitized histogram with larger size than H. As we will explain in Sect. 4.1, it is straightforward to optimally solve SLH_r based on our *LHO* algorithm.

3.2.1 Solutions to the *SLH* optimization problem satisfy the desired privacy property

The adversary cannot distinguish between a user A who has only visited nonsensitive locations and thus submits a non-sanitized histogram H_A, and a user B who has visited some sensitive locations and the algorithm has produced a sanitized histogram H'_B that is identical to H_A. This is because every possible sanitized histogram that the *LHO* algorithm can output is a valid histogram that could have legitimately been produced by a user. Note that, if there are histograms that cannot be produced by a legitimate user, *LHO* can be trivially adapted to never output such histograms. This adaptation is easy because all histograms are encoded as paths in a graph, so illegitimate histograms are also paths in the graph, referred to as illegitimate paths, and these histograms can be avoided by simply changing the shortest path finding algorithm to an algorithm that finds a shortest path which is not contained in a given subset of illegitimate paths [44].

3.3 The Target Resemblance problem: adversary model and formal definition

As discussed in the Introduction and in Sect. 2, for the Target Resemblance (*TR*) privacy notion the user specifies a target histogram H'' to resemble, a quality parameter ϵ and a privacy parameter c. The objective of the *TR* optimization problem is to create a sanitized histogram H' that is as similar as possible to H'', subject to the quality constraint $d_q(H, H') \leq \epsilon$. The privacy distance function that quantifies the notion of similarity is denoted $d_p(H', H'')$. If $d_p(H', H'') > c$, then H' is not acceptable, because it is not similar enough to the target.

The function $d_p(H', H'')$ is nonnegative, and it must decompose as a sum over bins, i.e., there must be a function p such that $d_p(H', H'') = \sum_{i \in [1,n]} p(H'[i], H''[i])$, using zeros to fill in missing location counts. In *TR*, privacy is maximum when $H' = H''$ ($d_p(H', H'') = 0$), because there is no better resemblance than being identical. Any function with these properties would be suitable as d_p (e.g., *JS*-divergence, or L_2-distance). We use *JS*-divergence as d_p in our experiments (see Sect. 5).

We can formulate the adversary model and the desired privacy property for this problem as follows: the adversary knows (I) the histogram H' that the user submits, (II) a target histogram H'', (III) a privacy distance function $d_p()$, and (IV) a privacy parameter c.

Upon receiving H', the adversary compares it to the target H'' in order to profile the user. For example, if an adversary wants to determine whether the user is a member of a particular ethnic/religious/social group, the target histogram is the histogram of a typical member of that group. Formally, the adversary makes this determination by comparing $d_p(H', H'')$ to c, i.e., by comparing the privacy distance between the user's submitted histogram H' and H'' to the privacy parameter c. If $d_p(H', H'') \leq c$, the adversary concludes that the user is a member of the group, otherwise they conclude that the user is not a member of the group. The adversary has no other background knowledge. In particular, the adversary does not know whether the user submitted their true histogram or the user submitted a modified histogram aiming to resemble a particular target histogram. The adversary succeeds if they conclude that the user is not a member of the group, i.e., $d_p(H', H'') > c$.

The desired privacy property is the negation of the adversary's success criterion. In TR, the desired privacy property is $d_p(H', H'') \leq c$.

We formally define the corresponding optimization problem as follows:

Problem 3.2 (Target Resemblance (*TR*)) Given two histograms $H, H'' \in \mathcal{H}_{n,N}$, a privacy distance function $d_p()$, a privacy parameter c, a quality distance function $d_q()$, and maximum quality loss threshold $\epsilon \geq 0$, construct a sanitized histogram $H' \in \mathcal{H}_{n,N}$ that

$$\begin{aligned} \underset{H' \in \mathcal{H}_{n,N}}{\text{minimizes}} \quad & d_p(H', H'') \\ \text{subject to} \quad & d_q(H, H') \leq \epsilon. \end{aligned}$$

If the resulting H' is such that $d_p(H', H'') > c$, then it is impossible to achieve both the desired privacy property and the desired quality constraint.

Intuitively, the *TR* problem requires constructing a sanitized histogram H' of the same length and size with H and H'' that offers the best possible privacy by being as similar as possible to the target histogram H'' according to d_p, while incurring a quality loss at most ϵ according to d_q.

The function d_q is selected by the location-based application provider (recipient of the sanitized histogram) and is provided to the user together with an intuitive explanation of what different values of $d_q()$ mean for quality. For example, $d_q() \geq 0.8$ means "very low quality," $0.6 \leq d_q() \leq 0.8$ means "low quality" etc., where "quality" refers to the quality of the application response (e.g., recommendation) that the user receives. Then, in the spirit of user-centric (or personalized) privacy [19,51], the user uses the above explanation by the provider to choose a value of ϵ that corresponds to his/her tolerance for quality loss.

The problem requires the user to specify the target histogram H''. However, some users may not possess sufficient knowledge to perform this task, even though they want to resemble a person with certain characteristics (e.g., a wealthy person). In these cases, H'' can be constructed as follows. The user chooses a target probability distribution h'' from a repository of probability distributions that are constructed by domain experts and labeled accordingly (e.g., a distribution corresponding to a "wealthy" profile, a "tourist" profile, a "healthy person" profile [22,56]), in the same way that experts compile, e.g., adblock filters (lists of URLs to block) or lists of virus signatures for antivirus software. To choose one of these profiles, the user looks for a label that they want to resemble. This setup is very similar to other papers in the literature [1,13].

Note that the distribution h'' may be defined on a different set of locations from the user's histogram H, in which case both are expanded to cover all locations in either h'' or H, with zero values for the new locations. Then, each entry $h''[i]$ is multiplied by the size N of the user's histogram H to create the target histogram $H''[i]$ (see Sect. 4.2). So, in effect, H'' is the expected histogram by a hypothetical user that picks N locations from the distribution h''. By the above construction, H'' and H are of the same length n and size N, but note that H'' may not have integer counts, because $H''[i] = N \cdot h''[i]$ is not necessarily an integer. Strictly speaking, this violates the requirement of histograms to have integer counts, but that is not a problem for our methods, because the privacy distance functions do not need integer arguments. However, we do require the histogram H' that the algorithms output to have integer counts.

Theorem 3.2 *The TR problem is weakly NP-hard.*

Proof See Appendix (Sect. A.2). □

Clearly, the *TR* problem requires constructing a sanitized histogram H' with the same size as H and H''. That is, it assumes that the desirable target histogram H'' has the same counts as H, but these counts are distributed differently from H. However, it is also possible to relax this assumption. This leads to a variation of the *TR* problem, referred to as $TR_{|H''|_1}$, which instead requires the sanitized histogram H' to only have the same size as H'', while it can be different from the size of H. It is straightforward to optimally (resp., heuristically) solve $TR_{|H''|_1}$ based on our *RO* algorithm (resp., based on our *RH* heuristic) (see Sect. 4.2).

3.3.1 Solutions to the *TR* optimization problem satisfy the desired privacy property

The TR problem tries to minimize $d_p(H', H'')$, while satisfying the quality constraint $d_q(H, H') \leq \epsilon$. Of course, a particular choice of ϵ affects privacy. If ϵ is low, an algorithm for the *TR* problem may output an H' that is the same or very similar to H, because all histograms that satisfy the specified quality constraint are close to H. Then, the user has to decide whether this H' is safe to release.

Given the privacy parameter c, it is not safe to release H' when $d_p(H', H'') > c$. If $d_p(H', H'') > c$, the user will decide not to release any histogram at all. Alternatively, the user may want to re-run the algorithm with a larger ϵ, i.e., to sacrifice more quality in order to achieve the privacy requirement.

The user's decision may depend on the intuitive meaning of the function used for d_p. For example, if d_p is Pearson χ^2 and the target H'' models a "wealthy" user, then $d_p(H', H'')$ quantifies how much more likely it is that H' has been produced by a user who follows the "wealthy" profile compared to any other profile.[2] Thus, if this likelihood ratio exceeds c, then the user may not want to release that H'.

It is also trivial to exclude solutions with $d_p(H', H'') > c$ by modifying our methods to disregard such solutions and terminate if no solution exists. In conclusion, the user either submits a histogram that satisfies the privacy property, or nothing at all.

3.4 The Target Avoidance problem

As mentioned above, Target Avoidance (*TA*) is a variant of the Target Resemblance (*TR*) problem, which we briefly discuss below. The algorithms for the *TA* problem are very similar to those for *TR* and are omitted; for details see [33].

Problem 3.3 (Target Avoidance (*TA*)) Given two histograms $H, H'' \in \mathcal{H}_{n,N}$, a privacy distance function $d_p()$, a privacy parameter c, a quality distance function $d_q()$, and maximum quality loss threshold $\epsilon \geq 0$, construct a sanitized histogram $H' \in \mathcal{H}_{n,N}$ that

$$\begin{aligned} \underset{H' \in \mathcal{H}_{n,N}}{\text{maximizes}} \quad & d_p(H', H'') \\ \text{subject to} \quad & d_q(H, H') \leq \epsilon. \end{aligned}$$

If the resulting H' is such that $d_p(H', H'') < c$, then it is impossible to achieve both the desired privacy property and the desired quality constraint.

Intuitively, the *TA* problem requires constructing a sanitized histogram H' of the same length and size with H and H''. The sanitized histogram must offer the best possible privacy

[2] We refer the interested reader to statistics textbooks for more details [25].

by being as dissimilar as possible to the target histogram H'' according to d_p, while incurring a quality loss at most ϵ according to d_q. The threshold ϵ and target histogram H'' are specified by the user based on their preferences. For example, the user can set H'' to H, in order to avoid H itself, or to a part of H that contains the locations that characterize an undesirable profile (e.g., frequent visits to airports) or are frequented by a certain ethnic minority (which may help infer that an individual belongs to the minority). The user could also choose H'' with the help of domain experts, as in the TR problem.

In terms of an adversary model, the adversary has the same knowledge as in *TR* and they succeed if $d_p(H', H'') < c$. If the algorithm does not produce an H'' such that $d_p(H', H'') \geq c$, then the user can either not submit any histogram at all, or the user may want to re-run the algorithm with a larger ϵ. The proof that the *TA* problem leads to a solution satisfying the desired privacy property is similar to that for *TR* (omitted).

Theorem 3.3 *The TA problem is weakly NP-hard.*

Proof Omitted (see [33]). □

The *TA* problem is very similar to the *TR* problem. This is established through a reduction from *TA* to *TR* that is given in Appendix (Sect. A.3). There is also a variation of *TA*, referred to as $TA_{|H''|_1}$, which requires the sanitized histogram H' to have the same size as H'', but not necessarily as H. Again, our methods can easily deal with this variation.

4 Algorithms

Since the *SLH*, *TR*, and *TA* problems are weakly NP-hard, it is possible to design pseudopolynomial algorithms to optimally solve them. Such algorithms run in polynomial time in the numerical value of the input [41]. We present optimal algorithms based on shortest path problems for the *SLH* and *TR* problems. In addition, we present a heuristic algorithm for the *TR* problem, which is more efficient than the optimal algorithm by two orders of magnitude and finds solutions of comparable quality. Furthermore, we explain how our methods can deal with the variations SLH_r and $TR_{|H''|_1}$ of the *SLH* and *TR* problem, respectively.

4.1 *LHO*: an optimal algorithm for *SLH*

This section presents Location Hiding Optimal (*LHO*), which optimally solves the *SLH* problem. Before presenting *LHO*, as motivation, we consider a simple algorithm which distributes the counts of the sensitive location(s) to the nonsensitive bin(s) proportionally to the counts of the nonsensitive bins. Thus, it aims to construct an H' by initializing it to H and then increasing the count of each nonsensitive bin $H'[i]$ by $x[i] = H[i] \cdot \frac{\sum_{i \in L'} H[i]}{\sum_{i \in L \setminus L'} H[i]} = H[i] \cdot \frac{K}{N-K}$, while assigning 0 to each sensitive bin. While intuitive, this algorithm fails to construct an H', for a given histogram H and distance function d_q, when $x[i]$ is not an integer, and also it may lead to solutions with large $d_q(H, H')$ (i.e., low data utility), as it does not take into account the input distance function d_q.

We now discuss the *LHO* algorithm. Without loss of generality, we assume that the nonsensitive locations correspond to the first $n - |L'|$ bins of the original histogram $H = (f(L_1), \ldots, f(L_n))$, while the remaining $|L'|$ bins correspond to the sensitive locations. The total count of sensitive locations in H is $K = \sum_{L_i \in L'} f(L_i)$. *LHO* must move (redistribute) these counts into the nonsensitive bins, while minimizing the quality error $d_q()$.

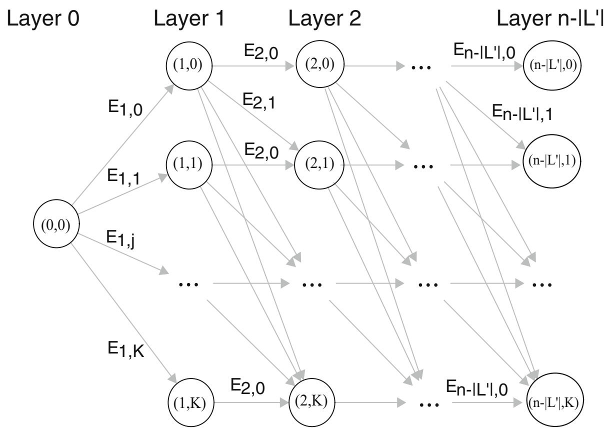


Fig. 4 Search space graph G_{LHO} for the Sensitive Location Hiding problem. Layer 0 contains the node $(0, 0)$ and each of the layers 1 to $n - |L'|$ contains $K + 1$ nodes. Each edge connects nodes of consecutive layers and has a weight equal to the error $E_{i+1,k}$, where $i + 1$ is the layer of the end node of the edge and k is the count of sensitive locations. $E_{i+1,k}$ represents the impact of redistributing (i.e., adding) k counts into the $i + 1$ bin of the sanitized histogram H', which is initialized to the original histogram H. That is, $E_{i+1,k} = q(H[i + 1], H'[i + 1] + k)$. The missing nodes and edges are denoted with "..."

The *LHO* algorithm is founded on the following observation: the optimal way of redistributing counts to each nonsensitive bin of H' corresponds to a shortest path between two specific nodes of a *search space graph* $G_{LHO}(V, E)$, where V and E are the set of nodes and set of edges of G_{LHO}, respectively. In the following, we discuss the construction of G_{LHO} and the correspondence between this shortest path and the solution to the *SLH* problem. Then, we discuss the *LHO* algorithm.

G_{LHO} is a multipartite directed acyclic graph (DAG) (see Fig. 4) such that:

- It contains $n - |L'| + 1$ layers of nodes. Layer 0 comprises a single node, and layers $1, \ldots, n - |L'|$ comprise $K + 1$ nodes each. Each layer $1, \ldots, n - |L'|$ corresponds to a nonsensitive bin.
- The single node in layer 0 is labeled $(0, 0)$, and each node in layer $i \in [1, n - |L'|]$ is labeled (i, j), where j denotes the redistribution (i.e., addition) of j counts to bins 1 up to and including i of the sanitized histogram. We may refer to nodes of G_{LHO} using their labels.
- There is an edge $((i, j), (i + 1, j + k))$ from node (i, j) to node $(i + 1, j + k)$, for each $i \in [0, n - |L'| - 1]$, where $k \geq 0$, $j + k \leq K$. That is, each node labeled (i, j) is connected to every node in the following layer $i + 1$ that corresponds to a count of at least j.
- Each edge $((i, j), (i + 1, j + k))$ is associated with a weight equal to the error $E_{i+1,k} = q(H[i + 1], H[i + 1] + k)$. The error $E_{i+1,k}$ quantifies the impact on quality that is incurred by redistributing (i.e., adding) k counts into bin $i + 1$.

Let P be a path comprised of nodes $(0, 0)$, $(1, k_1)$, ..., $(n - |L'|, k_{n-|L'|})$ of G_{LHO}. The properties below easily follow from the construction of G_{LHO}:

(I) The path P corresponds to an addition of $k_i - k_{i-1}$ counts to the i-th bin of the histogram, for each $i \in [1, n - |L'|]$, where $k_0 = 0$.
(II) The *length* of P is equal to the total weight $E_{1,k_1} + \ldots, E_{n-|L'|,k_{n-|L'|}}$ of the edges in P. This total weight is the total quality loss incurred by the allocation corresponding to P.

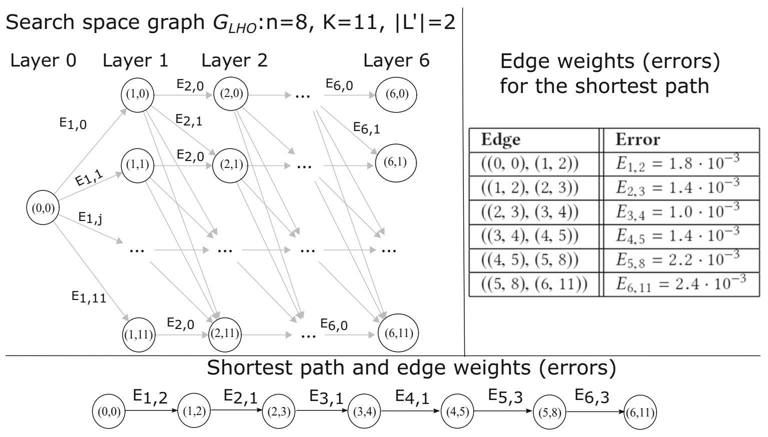
Thus, the path P corresponds to a sanitized histogram H' whose first $n-|L'|$ bins have counts $H[i]+(k_i-k_{i-1})$, $i=1,\ldots,n-|L'|$, and $d_q(H,H')$ is equal to $\sum_{i=1}^{n-|L'|} q(H[i],H[i]+k_i-k_{i-1})$. For example, the path comprised of nodes $(0,0)$, $(1,K),\ldots,(n-|L'|,K)$ in Fig. 4 corresponds to a sanitized histogram H' in which all K sensitive counts have been moved to the first bin. The quality loss in this case is just $d_q(H,H')=q(H[1],H[1]+K)$, as all other bins have the same counts in both H and H'.

Conversely, each possible allocation of the K sensitive counts into nonsensitive bins corresponds to a path between the nodes $(0,0)$ and $(n-|L'|,K)$ of G_{LHO}, which represents a feasible solution to the *SLH* problem. Therefore, the shortest path between the nodes $(0,0)$ and $(n-|L'|,K)$) of G_{LHO} (i.e., the path with the minimum length $E_{1,k_1}+\ldots,E_{n-|L'|,K}$; ties are broken arbitrarily) represents a sanitized histogram $H'=(H[1]+(k_1-k_0),\ldots,H[n-|L'|]+(K-k_{n-|L'|-1}),0,\ldots,0)$, which is the optimal solution to *SLH*. This is because H' has minimum $d_q(H,H')$, the same size with H, and a zero count for each sensitive location.

We now present the pseudocode of the *LHO* algorithm. In step 1, the algorithm constructs the search space graph G_{LHO}. In step 2, the algorithm finds a shortest path between the nodes $(0,0)$ and $(n-|L'|,K)$ of G_{LHO}. In step 3, the sanitized histogram H' corresponding to the shortest path (i.e., the optimal solution to the *SLH* problem) is created and, last, in step 4, H' is returned.

```
Algorithm: LHO (Location Hiding Optimal)
Input: Histogram $H$, set of sensitive locations $L'$, quality distance function $d_q$
Output: Sanitized histogram $H'$
1 Construct the search space graph $G_{LHO}$
2 $((0,0),(1,k_1),\ldots,(n-|L'|,K)) \leftarrow$ the shortest path from node $(0,0)$ to node
          $(n-|L'|,K)$ in $G_{LHO}$
3 $H' \leftarrow (H[1]+k_1, H[2]+(k_2-k_1),\ldots,H[n-|L'|]+(K-k_{n-|L'|-1}),0,\ldots,0)$
4 return $H'$
```

Example 4.1 *LHO* is applied to the histogram $H=(7,2,3,2,13,12,8,3)$ in Fig. 1a. The set of sensitive locations L' contains the locations g and h with counts 8 and 3, respectively, and the quality distance function d_q is *JS*-divergence. In step 1, the algorithm constructs the search space graph in Fig. 5. The graph has $n-|L'|+1=7$ layers of nodes, where $n=8$ is the length of H and $|L'|=2$ is the number of sensitive locations. Layer 0 contains the node $(0,0)$ and each other layer contains $K+1=12$ nodes, where $K=11$ is the total count of sensitive locations in H. Each node in layers $1,\ldots,6$ is labeled (i,j); i denotes the layer of the node and j denotes the counts of sensitive locations that are redistributed into bins $1,\ldots,i$. For example, the node $(6,11)$ denotes that all 11 counts of the sensitive locations are added into bins $1,\ldots,6$. In addition, there is an edge with weight $E_{i+1,k}$ between each node (i,j) and every node $(i+1,j+k)$, for each $k\in[0,K-j]$. The weight $E_{i+1,k}$ quantifies the increase to *JS*-divergence incurred by redistributing (i.e., adding) k counts of sensitive locations into bin $i+1$. For example, the node $(0,0)$ is connected to the nodes $(1,0),\ldots,(1,11)$, and the edge $((0,0),(1,2))$ has weight $E_{1,2}\approx 1.8\times10^{-3}$, because adding 2 counts into the first bin increases *JS*-divergence by approximately 1.8×10^{-3}. In step 2, *LHO* finds the shortest path from the node $(0,0)$ to the node $(6,11)$, shown in Fig. 5, and in step 3 it constructs the sanitized histogram $H'=(9,3,4,3,16,15,0,0)$ (see Fig. 1b) that corresponds to the shortest path. Note that j in the label (i,j) of each node in the shortest path corresponds to the counts of sensitive locations that are added into bins $1,\ldots,i$ in H'. Last, in step 4, H' is returned. □



Edge	Error
((0, 0), (1, 2))	$E_{1,2} = 1.8 \cdot 10^{-3}$
((1, 2), (2, 3))	$E_{2,3} = 1.4 \cdot 10^{-3}$
((2, 3), (3, 4))	$E_{3,4} = 1.0 \cdot 10^{-3}$
((3, 4), (4, 5))	$E_{4,5} = 1.4 \cdot 10^{-3}$
((4, 5), (5, 8))	$E_{5,8} = 2.2 \cdot 10^{-3}$
((5, 8), (6, 11))	$E_{6,11} = 2.4 \cdot 10^{-3}$

Fig. 5 Search space graph G_{LHO} for Example 4.1 (the missing nodes and edges are denoted with "…"), and shortest path along with its corresponding weights

The time complexity of *LHO* is $O\left((n-|L'|)\cdot K^2+S\right)$, where $(n-|L'|)\cdot K^2$ is the cost of constructing G_{LHO} (step 1) and S is the cost of finding the shortest path (step 2). Constructing G_{LHO} takes $O\left((n-|L'|)\cdot K^2\right)$ time, because G_{LHO} contains $O\left((n-|L'|)\cdot K\right)$ nodes and $O\left(K+1+(n-|L'|-1)\cdot\binom{K}{2}\right)=O\left((n-|L|)\cdot K^2\right)$ edges, and the computation of each edge weight $E_{i+1,k}$ takes $O(1)$ time, because it is computed by accessing a single pair of bins from H and H'. The cost S is determined by the shortest path algorithm (e.g., it is $O\left((n-|L|)\cdot K^2\cdot\log\left((n-|L'|)\cdot K^2\right)\right)$ for Dijkstra's algorithm with binary heap [47]).

Note, the variation SLH_r in Sect. 3.2 can be optimally solved by applying *LHO* with $K=r$.

4.2 Optimal algorithm for Target Resemblance

In this section, we model and solve the *TR* problem as a constrained shortest path problem on a specially constructed search space graph G_{TR}. It follows immediately that the *TA* problem can be seen as a *longest* path problem on the same graph. Because the graph is a DAG, computing longest and shortest paths has the same complexity [47]: by visiting the graph nodes in breadth-first search order, we can simply keep track of the shortest (or longest) path to each node. We can even solve the two problems in one pass. We focus on the *TR* problem, which we solve optimally with the Resemblance Optimal (*RO*) algorithm.

In the following, we discuss the construction of G_{TR} and then provide the pseudocode of *RO*.

From the histogram H and the distance functions d_p and d_q, we construct a multipartite DAG $G_{TR}=(V,E)$, as follows (see also Fig. 6):

- There are $n\cdot(N+1)+1$ nodes in V, where n and N are the length and the size of the histogram H, respectively.
- The nodes are arranged in layers $0,1,\ldots,n$, with layer 0 having a single node and layers $1,\ldots,n$ having $N+1$ nodes each. Layer $i\in[1,n]$ corresponds to bin i (location L_i) in the histogram. Node $j\in[0,N]$ in layer i corresponds to the allocation of a total of j frequency counts to histogram bins 1 up to and including i.

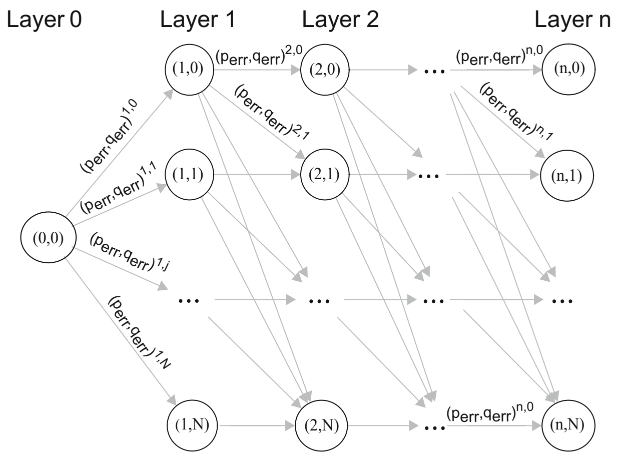

Layer 0
Layer 1
Layer 2
Layer n
(0,0)
(1,0)
(1,1)
(1,N)
(2,0)
(2,1)
(2,N)
(n,0)
(n,1)
(n,N)
$(p_{err},q_{err})^{1,0}$
$(p_{err},q_{err})^{1,1}$
$(p_{err},q_{err})^{1,j}$
$(p_{err},q_{err})^{1,N}$
$(p_{err},q_{err})^{2,0}$
$(p_{err},q_{err})^{2,1}$
$(p_{err},q_{err})^{n,0}$
$(p_{err},q_{err})^{n,1}$
$(p_{err},q_{err})^{n,0}$

Fig. 6 Search space graph G_{TR} for the Target Resemblance problem. Layer 0 is an auxiliary layer that just contains the node $(0, 0)$. Layer $i = 1, \ldots, n$ corresponds to bin i of the sanitized histogram, and node (i, j) corresponds to allocating $j = 1, \ldots, N$ counts to bins 1 up to and including i. A path from $(0, 0)$ to (n, N) completely defines an allocation of N counts to n bins. The weight of the edge from (i, j) to $(i + 1, j + k)$ is the privacy and quality error of allocating exactly k counts to bin $i + 1$. As these errors are additive, the admissible paths are those whose total q-length is less than the threshold ϵ. Among them, the p-shortest path from $(0, 0)$ to (n, N) corresponds to the optimal solution to *TR*, because it also has q-length at most ϵ

- The single node in layer 0 is labeled $(0, 0)$, and each node j in every other layer i is labeled (i, j). We may refer to nodes of G_{TR} using their labels.
- The edges in E go from each node (i, j) to each node $(i + 1, j + k)$, $k \geq 0$, $j + k \leq N$, i.e., to each node in the following layer that has a frequency count at least equal to j.
- The weight of an edge from (i, j) to $(i + 1, j + k)$ is the pair $(p_{\text{err}}, q_{\text{err}})^{i+1,k}$ of the privacy and quality errors of allocating exactly k counts to bin $i + 1$ of the sanitized histogram: $p_{\text{err}} = p(k, H''[i + 1])$, $q_{\text{err}} = q(H[i + 1], k)$. The p-length of a path is the sum of its p_{err} weights. We will refer to the path with the minimum p-length as the p-shortest path. Similarly, the q-length of a path is the sum of its q_{err} weights.

At this point, note two important differences between the edge weights of G_{TR} and G_{LHO} (Sect. 4.1): first, and most obvious, the edge weights in G_{TR} are pairs of (privacy error, quality error), whereas in *SLH* the weights are quality errors. Second, in G_{TR} the weight of edge from (i, j) to $(i + 1, j + k)$ corresponds to setting $H'[i + 1]$ exactly equal to k, whereas in G_{LHO} that edge weight would correspond to setting $H'[i + 1]$ equal to $H[i + 1] + k$.

From the construction of G_{TR}, it follows that there is a $1 - 1$ correspondence between a sanitized histogram $H' \in \mathcal{H}_{n,N}$ and a path from $(0, 0)$ to (n, N) in G_{TR}. Therefore, to solve the *TR* problem, we need to find the path from $(0, 0)$ to (n, N) with minimum p-length among the paths whose q-length is at most ϵ. Then, it is straightforward to construct the histogram from the path.

We now provide the pseudocode of the *RO* algorithm. We assume that the preprocessing needed to construct H'' from h'' is done before the actual algorithm runs, and also H and H'' have been expanded to be defined on the same set of locations, if needed (see Sect. 3.3 for details on h''). Also, for the moment, we assume that d_q takes nonnegative integer values.

In step 1, *RO* constructs the graph G_{TR}. In steps 2–6, the algorithm iterates over each node v of the graph and associates with it a vector $\mathcal{V}_v$, indexed by all possible values of d_q. The elements of $\mathcal{V}_v$ are initialized to 0 for node $(0, 0)$, and to ∞ for any other node of G_{TR}. Next,

```
Algorithm: RO (Resemblance Optimal)
Input: Histogram $H$, target histogram $H''$, privacy distance function $d_p$, quality distance function
       $d_q$, maximum quality loss threshold $\epsilon$
Output: Sanitized histogram $H'$
1  Construct the graph $G_{TR}$
2  foreach node $v$ in $G_{TR}$ do
3      if the label of $v$ is $(0,0)$ then
4          Associate the node $v$ with a vector $\mathcal{V}_v$ s.t. $\mathcal{V}_v[k]=0$, for each integer $k \in [0,\epsilon]$
5      else
6          Associate the node $v$ with a vector $\mathcal{V}_v$ s. t. $\mathcal{V}_v[k]=\infty$, for each integer $k \in [0,\epsilon]$
7  foreach node $v$ in $G_{TR}$ in increasing lexicographic order starting from node $(1,0)$ do
8      foreach element $k$ of $\mathcal{V}_v$ do
9          $\mathcal{V}_v[k]=\min_{(u,v)\in E, q_{err}(u,v)\le k}\{\mathcal{V}_u[k-q_{err}(u,v)]+p_{err}(u,v)\}$
10 $((0,0),(1,j_1),\ldots,(n,N))$ ← the shortest path from node $(0,0)$ to node $(n,N)$ in
                                   $G_{TR}$. Its $p$-length is equal to the minimum element of $\mathcal{V}_v$
                                   for node $v=(n,N)$.
11 $H' \leftarrow (j_1, j_2-j_1,\ldots,N-j_{N-1})$
12 return $H'$
```

in steps 7–9, *RO* iterates over the nodes of G_{TR} in increasing lexicographic order, starting from node $(1,0)$, and for each node v it updates all the elements of $\mathcal{V}_v$. Each element $\mathcal{V}_v[k]$ is updated using the following dynamic programming equation:

$$\mathcal{V}_v[k]=\min_{(u,v)\in E, q_{\mathrm{err}}(u,v)\le k}\{\mathcal{V}_u[k-q_{\mathrm{err}}(u,v)]+p_{\mathrm{err}}(u,v)\}. \tag{4.1}$$

The element $\mathcal{V}_v[k]$ is equal to the p-length of the p-shortest path from $(0,0)$ to node v with q-length exactly equal to k. Thus, as explained above, this path is a feasible solution to the *TR* problem, and so the optimal solution to *TR* is the p-shortest path from $(0,0)$ to (n,N) (i.e., the path corresponding to the minimum element of the vector $\mathcal{V}_{(n,N)}$). The nodes of this path are found in step 10, and its corresponding histogram H' is constructed in step 11.

We now consider the general case in which the values of q-length of a path from $(0,0)$ to (n,N) are not necessarily integer. We first show that the q-length of this path is polynomial in N, in Theorem 4.1 below. Then, we show that the number of values of q-length for all paths is polynomial in N, which implies that these values are not too many to store in the vectors $\mathcal{V}_u$.

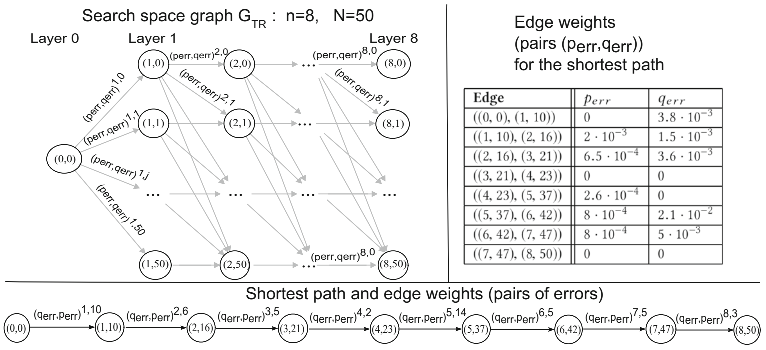
Theorem 4.1 *The q-length of a path from $(0,0)$ to (n,N) can only take a polynomial (in N) number of values.*

Proof The number of possible allocations of N elements to n bins, where some of the bins may be left empty, is equal to the number of n-tuples of nonnegative integers $f_1,\ldots,f_N$ that sum to N. Such tuples are called *weak compositions* of N into n parts (*weak*, because zeros are allowed), and their total number is

$$\binom{N+n-1}{n-1}=\frac{(N+1)\ldots(N+n-1)}{(n-1)!}, \tag{4.2}$$

which is a polynomial in N [6]. □

Equation 4.2 gives all possible q-lengths for a path from $(0,0)$ to (n,N). We also need to keep *intermediate* q-length values in the vectors $\mathcal{V}_v$, i.e., q-lengths for paths from $(0,0)$ to nodes in any layer $1,\ldots,n$. But each intermediate allocation has at most as many q-length values as the final one, because an intermediate allocation allocates at most N elements to


Search space graph G_{TR} : n=8, N=50
Layer 0
Layer 1
Layer 8
(0,0)
(1,0)
(1,1)
(1,50)
(2,0)
(2,1)
(2,50)
(8,0)
(8,1)
(8,50)
$(p_{err},q_{err})^{1,0}$
$(p_{err},q_{err})^{1,1}$
$(p_{err},q_{err})^{1,j}$
$(p_{err},q_{err})^{1,50}$
$(p_{err},q_{err})^{2,0}$
$(p_{err},q_{err})^{2,1}$
$(p_{err},q_{err})^{8,0}$
$(p_{err},q_{err})^{8,1}$
Edge weights
(pairs (p_{err},q_{err}))
for the shortest path
Shortest path and edge weights (pairs of errors)
(1,10)
(2,16)
(3,21)
(4,23)
(5,37)
(6,42)
(7,47)
$(q_{err},p_{err})^{1,10}$
$(q_{err},p_{err})^{2,6}$
$(q_{err},p_{err})^{3,5}$
$(q_{err},p_{err})^{4,2}$
$(q_{err},p_{err})^{5,14}$
$(q_{err},p_{err})^{6,5}$
$(q_{err},p_{err})^{7,5}$
$(q_{err},p_{err})^{8,3}$

Edge	p_{err}	q_{err}
((0, 0), (1, 10))	0	$3.8 \cdot 10^{-3}$
((1, 10), (2, 16))	$2 \cdot 10^{-3}$	$1.5 \cdot 10^{-3}$
((2, 16), (3, 21))	$6.5 \cdot 10^{-4}$	$3.6 \cdot 10^{-3}$
((3, 21), (4, 23))	0	0
((4, 23), (5, 37))	$2.6 \cdot 10^{-4}$	0
((5, 37), (6, 42))	$8 \cdot 10^{-4}$	$2.1 \cdot 10^{-2}$
((6, 42), (7, 47))	$8 \cdot 10^{-4}$	$5 \cdot 10^{-3}$
((7, 47), (8, 50))	0	0

Fig. 7 Search space graph G_{TR} for Example 4.2 (the missing nodes and edges are denoted with "…"), and shortest path along with its corresponding edge weights

at most n bins. As there are n stages of intermediate allocations, we have in total at most $n \cdot \binom{N+n-1}{n-1}$ values, i.e., a polynomial in N.

Example 4.2 *RO* is applied to the histogram $H = (7, 2, 3, 2, 13, 12, 8, 3)$ in Fig. 1a, using the target histogram $H'' = (10, 8, 6, 2, 13, 4, 4, 3)$ in Fig. 1c, *JS*-divergence as the quality distance function d_q and the privacy distance function d_p, and $\epsilon = 0.05$. In step 1, the algorithm constructs the search space graph in Fig. 7. The graph has $n + 1 = 9$ layers of nodes, where $n = 8$ is the length of H. Layer 0 contains the node $(0, 0)$ and each other layer contains $N + 1 = 51$ nodes, where $N = 50$ is the size of H. Each node in layers $1, \ldots, 8$ is labeled (i, j); $i \in [1, 8]$ denotes the layer of the node and corresponds to bin i, while $j \in [0, 50]$ denotes the counts allocated to bins $1, \ldots, i$. For example, the node $(8, 50)$ denotes that all 50 counts of H are allocated to bins $1, \ldots, 8$. In addition, there is an edge from each node (i, j) to every node $(i + 1, j + k)$, for each $k \in [0, 50 - j]$. The edge weight is a pair $(p_{\text{err}}, q_{\text{err}})$, where the privacy error p_{err} (respectively, quality error q_{err}) quantifies the error with respect to *JS*-divergence that is incurred by allocating k counts to bin $i+1$ of the sanitized histogram (see Fig. 7). For example, the node $(0, 0)$ is connected to the nodes $(1, 0), \ldots, (1, 50)$, and the edge $((0, 0), (1, 10))$ has $(p_{\text{err}}, q_{\text{err}})^{1,10} = (0, 3.8 \times 10^{-3})$, incurred by allocating 10 counts to the first bin. In steps 2–9, *RO* computes the vector $\mathcal{V}_u$ for each node u. In step 10, the algorithm finds the shortest path from node $(0, 0)$ to node $(8, 50)$ with $q_{\text{err}} \leq \epsilon$ (see Fig. 7), and in step 11 it constructs the sanitized histogram $H' = (10, 6, 5, 2, 14, 5, 5, 3)$ that corresponds to the shortest path (see Fig. 1d). Note that j in the label (i, j) of each node in the shortest path corresponds to the counts of sensitive locations that are allocated to bins $1, \ldots, i$ in H'. Last, in step 12, H' is returned. □

The time complexity of the *RO* algorithm is $O((n \cdot N)^2 \cdot \binom{N+n-1}{n-1})$. The total cost is the sum of the cost of constructing G_{TR} and of finding the constrained shortest path from $(0, 0)$ to (n, N).

The construction of G_{TR} takes $O(n \cdot N^2)$ time. This is because the algorithm constructs $O(n \cdot (N + 1) + 1) = O(n \cdot N)$ nodes, each of which has $O(N)$ outgoing edges, for a total of $O(n \cdot N^2)$ edges. Note also that the computation of each edge weight takes $O(1)$ time. The cost of computing the shortest path is $O((n \cdot N)^2 \cdot \binom{N+n-1}{n-1})$. This is because it requires (I) constructing a vector $\mathcal{V}_v$ with $O(n \cdot \binom{N+n-1}{n-1})$ entries, for each node v of the $O(n \cdot N)$ nodes of G_{TR}, which takes $O(n^2 \cdot N \cdot \binom{N+n-1}{n-1})$ time, and (II) updating each entry of $\mathcal{V}_v$

once, which takes $O(N)$ time per node since there are $O(N)$ incoming edges to each node (see Eq. 4.1), for a total of $O\left((n \cdot N)^2 \cdot \binom{N+n-1}{n-1}\right)$ across all nodes.

Note, the variation $TR_{|H''|_1}$ in Sect. 3.3 can be optimally solved by using *RO* to allocate $|H''|_1$ counts instead.

4.3 Heuristic for Target Resemblance

Our heuristic *RH* for the Target Resemblance problem works in a greedy fashion to avoid the cost of constructing and searching the search space graph.

```
Algorithm: RH (Resemblance Heuristic)
Input: Histogram $H$, target histogram $H''$, privacy distance function $d_p$, quality distance funct. $d_q$,
       quality thr. $\epsilon$
Output: Sanitized histogram $H'$
1  $SrcBins \leftarrow \{i \text{ such that } H[i] > H''[i]\}$
2  $DstBins \leftarrow \{i \text{ such that } H[i] < H''[i]\}$
3  $H' \leftarrow H$
4  $\epsilon_{rem} \leftarrow \epsilon$   // Remaining quality budget
5  while $SrcBins \neq \emptyset$ do
       // Perform the best move on $H'$
6      $(H', Opt\Delta d_q) \leftarrow$ BestMove$(H', H'', SrcBins, DstBins, \epsilon_{rem})$
       // Exit if the remaining budget is exhausted
7      if $Opt\Delta d_q = -1$ then
8          break
       // Update the remaining budget
9      $\epsilon_{rem} \leftarrow \epsilon_{rem} - Opt\Delta d_q$
       // Update the set of source and dest. bins
10     $SrcBins \leftarrow \{i \text{ such that } H'[i] > H''[i]\}$
11     $DstBins \leftarrow \{i \text{ such that } H'[i] < H''[i]\}$
12 return $H'$
```

The main idea in *RH* is to try to greedily reduce the differences in the counts of corresponding bins between H and H''. As can be seen in the pseudocode (steps 1 and 2), *RH* identifies *source bins*, i.e., bins in H with more counts in H than in H'', and *destination bins*, bins with fewer counts in H than in H''. Bins with equal counts in H and H'' are ignored. Then, in steps 3 and 4, H' is initialized to the original histogram H and the remaining quality budget ϵ_{rem} to the quality threshold ϵ. In steps 5 and 6, *RH* moves some counts from a source bin to a destination bin using a function $BestMove$.

As can be seen in the pseudocode of $BestMove$ (steps 4–6), the function searches all possible ways ("moves") to move k counts from a source bin i to a destination bin j. For each move, $BestMove$ computes the privacy effect Δd_p and the quality effect Δd_q (steps 10 and 11), and it selects the move that maximizes the ratio $\frac{\Delta d_p}{\Delta d_q}$, subject to the constraint that Δd_q cannot exceed the remaining quality budget ϵ_{rem} (steps 12–15).[3] The rationale is to

[3] Note that $\Delta d_q > 0$, because Δd_q is the sum of two positive terms (in square brackets): $\Delta d_q = d_q(H'_{tmp}, H) - d_q(H', H) = q(H'_{tmp}[i], H[i]) + q(H'_{tmp}[j], H[j]) - q(H'[i], H[i]) - q(H'[j], H[j]) = \left[q(H'[i]-k, H[i]) - q(H'[i], H[i])\right] + \left[q(H'[j]+k, H[j]) - q(H'[j], H[j])\right]$. The first term is positive: bin i is a source bin, which means $H'[i] \leq H[i]$. But $H'[i] - k < H'[i]$, so the distance $q(H'[i]-k, H[i])$ is larger than $q(H'[i], H[i])$. Similarly, the second term is positive, because bin j is a destination bin.

```
Function: BestMove
Input: Sanitized histogram $H'$, target histogram $H''$, Set of source bins $SrcBins$, Set of destination
        bins $DstBins$, Remaining budget $\epsilon_{rem}$, Privacy distance function $d_p$, Quality distance
        function $d_q$
Output: Sanitized histogram $H'$ after performing the best move, difference in $d_q$ incurred by the best
         move
1  $MaxRatio \leftarrow 0$
2  $Opt\Delta d_q \leftarrow -1$
3  $H'_{BestMove} \leftarrow H'$
4  foreach bin $i$ in $SrcBins$ do
5      foreach bin $j \neq i$ in $DstBins$ do
6          foreach $k \in [1, H'[i]]$ do
               // Try moving $k$ counts from a source bin $H'[i]$ to a
                  destination bin $H'[j]$
7              $H'_{tmp} \leftarrow H'$
8              $H'_{tmp}[i] \leftarrow H'[i] - k$
9              $H'_{tmp}[j] \leftarrow H'[j] + k$
10             $\Delta d_p \leftarrow \left| d_p(H'_{tmp}, H'') - d_p(H', H'') \right|$
11             $\Delta d_q \leftarrow d_q(H'_{tmp}, H) - d_q(H', H)$
               // Store the best sanitized histogram so far, its ratio
                  and remaining budget
12             if $\frac{\Delta d_p}{\Delta d_q} > MaxRatio$ and $\Delta d_q < \epsilon_{rem}$ then
13                 $H'_{BestMove} \leftarrow H'_{tmp}$
14                 $MaxRatio \leftarrow \frac{\Delta d_p}{\Delta d_q}$
15                 $Opt\Delta d_q \leftarrow \Delta d_q$
16 $H' \leftarrow H'_{BestMove}$
17 return $(H', Opt\Delta d_q)$
```

prioritize moves with a large improvement in privacy Δd_p and a small reduction in quality Δd_q.

Next, in step 7, *RH* checks whether the remaining budget is exhausted. If it is, no more moves are performed (step 8). Otherwise, in steps 9–11, *RH* reduces the quality budget by $Opt\Delta_{d_q}$ (i.e., by the quality effect of the best move) and updates the sets of source and destination bins by no longer considering as source or destination bins any bins whose count has become equal to the corresponding bin in H''. Moves continue until the budget is exhausted or there are no more source/destination bins. Since moves cannot increase the count of a source bin nor increase the remaining quality budget, *RH* will always terminate.

The time complexity of *RH* is $O(n^3 \cdot N)$. This is because the loop in step 5 runs $O(n)$ times (once per source bin), and each time there is a cost of $O(n^2 \cdot N)$ incurred by $BestMove$. The cost of $BestMove$ is $O(n^2 \cdot N)$, because there are $O(n^2)$ source/destination bin pairs, and for each pair $O(N)$ temporary moves are performed. The time complexity analysis refers to the worst case. In practice, a histogram can be sanitized with a smaller number of moves (i.e., executions of $BestMoves$), and the heuristics scale well with respect to n. For example, in our experiments, the heuristics scale close to linearly with respect to n.

Last, we note that the *RH* heuristic can also directly deal with the variation $TR_{|H''|_1}$ of the *TR* problem (see Sect. 3.3). This is because *RH* does not pose any restriction on the size of H'', so H'' can have a different size than that of H.

Table 2 Characteristics of datasets

Dataset	# histograms	Mean of length n	Max. length n	Mean of size N
NYC	1083	40.28	139	209.99
TKY	2293	32.36	158	221.57

5 Evaluation

In this section, we evaluate our algorithms and heuristics in terms of effectiveness and efficiency. We do not compare against existing histogram sanitization methods, because they cannot be used to solve the problems we consider (see Sect. 6.2).

5.1 Setup and datasets

To calculate the loss in quality (utility) incurred by replacing the original histogram H with the sanitized histogram H', we compute the distance $d_q(H, H')$, where d_q is the Jensen–Shannon (JS) divergence (Sect. 3.1.1). Our algorithms can optimize other measures, leading to qualitatively similar results [33]. In addition, we measure how well sanitization preserves the quality of location recommendation.

Location recommendation suggests to a user, referred to as *active user* and denoted with α, a location that might interest them. We measure the impact of sanitization on recommendation quality based on the *recommendation error* [35], defined, for an active user α and a location L^{α}_{test}, as the difference between $f_\alpha(L^{\alpha}_{\text{test}})$, the user's *true* frequency of visits to L^{α}_{test}, and $r_\alpha(L^{\alpha}_{\text{test}})$, the frequency of visits as *predicted* by the recommendation algorithm. We use both the absolute error $|f_\alpha(L^{\alpha}_{\text{test}}) - r_\alpha(L^{\alpha}_{\text{test}})|$ and the square error $(f_\alpha(L^{\alpha}_{\text{test}}) - r_\alpha(L^{\alpha}_{\text{test}}))^2$. We compute recommendations based on the dataset of original user histograms, and then based on the dataset of sanitized histograms as follows: (I) each of these datasets is randomly partitioned into a training set D_{train} with 90% of the histograms and a test set D_{test} with 10% of the histograms, (II) the absolute (or square) recommendation error considering each user in D_{test} as α is computed, and (III) the errors are averaged to obtain two popular measures; *Mean Absolute Error* MAE and *Root Mean Square Error*. For the absolute error, $MAE(D_{\text{test}}) = \frac{1}{|D_{\text{test}}|}\sum_{(\alpha, L^{\alpha}_{\text{test}})} |f_\alpha(L^{\alpha}_{\text{test}}) - r_\alpha(L^{\alpha}_{\text{test}})|$, and *Root Mean Square Error* $RMSE(D_{\text{test}}) = \sqrt{\frac{1}{|D_{\text{test}}|}\sum_{(\alpha, L^{\alpha}_{\text{test}})} (f_\alpha(L^{\alpha}_{\text{test}}) - r_\alpha(L^{\alpha}_{\text{test}}))^2}$. For the square error, MAE and $RMSE$ are defined similarly.

All algorithms are implemented in Python and applied to the New York City (*NYC*) and Tokyo (*TKY*) datasets, which were also used in [39,63,64]. The datasets were downloaded from [62] and include long-term check-in data in New York city and Tokyo, collected from Foursquare from April 12, 2012 to February 16, 2013. Each record in the datasets contains a location that was visited by a user at a certain time and corresponds to a leaf in the Foursquare taxonomy (available at https://developer.foursquare.com/docs/resources/categories). There are in total 713 locations in the taxonomy, and on average each user visits fewer than 41 locations. For each dataset, we produce the input histograms for our algorithms by constructing one histogram H per user. The histogram H contains a count $f(L_i) > 0$ for every location L_i visited by the user. That is, H is constructed based on the user's values (location visits), which is line with histogram sanitization methods [2,16,23,26,45,61,68]. Table 2 shows the characteristics of *NYC* and *TKY*, and Table 3 shows the default values used in our experiments.

Table 3 Default values for each dataset w.r.t: length n, total frequency of sensitive locations K, number of sensitive locations $|L'|$, threshold ϵ, and histogram size N

Dataset	n	K	$\|L'\|$	ϵ	N
NYC	25	20	5	5×10^{-3}	100
TKY	35	20	5	5×10^{-3}	100

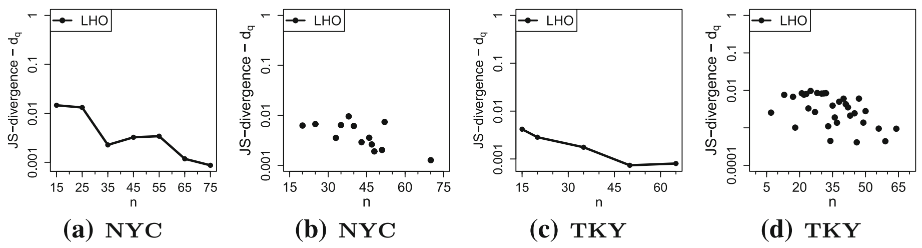


Fig. 8 JS-divergence versus length n: **a** Median JS-divergence for histograms of length n in *NYC*. **b** JS-divergence for each histogram with $K = 20$ in *NYC*. **c** Median JS-divergence for histograms of length n in *TKY*. **d** JS-divergence for each histogram with $K = 20$ in *TKY*

We also construct synthetic histograms by appending zeros to a histogram of length $n = 78$ and size $N = 192$ in *NYC* and to a histogram of $n = 99$ and $N = 642$ in *TKY*, and their length is up to 400, including the zero-frequency bins. We use the synthetic histograms to test the impact of length on the runtime performance of our methods. In total, we test the algorithms on approximately 3400 different histograms. All experiments ran on an Intel Xeon at 2.60GHz with 256GB of RAM.

5.2 Evaluation of the *LHO* algorithm

We evaluate the quality and runtime of *LHO* as a function of (I) n, the length of the original histogram, (II) K, the total frequency of sensitive locations, and (III) $|L'|$, the number of sensitive locations. We consider JS-divergence as $d_q()$ and an L' comprised of 5 sensitive locations selected randomly, unless stated otherwise.

5.2.1 Quality preservation for the *LHO* algorithm

Impact of histogram length n We show that JS-divergence decreases with n, in Fig. 8 (the y axis is in logarithmic scale). This is because there are more bins whose counts may increase: the space considered by *LHO* is larger and the change can be "smoothed" over more bins. Also, the JS-divergence scores are low, suggesting that sanitization preserves the distribution of nonsensitive locations fairly well.

Impact of total frequency of sensitive locations K We show that JS-divergence increases with K in Fig. 9 (the y-axes are in logarithmic scale). This is because there are more counts that must be redistributed into the bins of nonsensitive locations, and this incurs more distortion. Note, the JS-divergence scores are low, suggesting that the distribution of nonsensitive locations is preserved fairly well.

Impact of number of sensitive locations $|L'|$ We show that JS-divergence increases with $|L'|$, in Fig. 10a, b. This is because there are (I) more counts that need to be redistributed

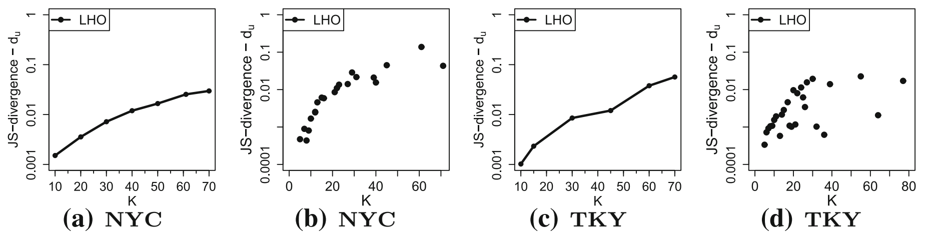


Fig. 9 JS-divergence versus total frequency of sensitive locations K: **a** Median JS-divergence for varying K in *NYC*. **b** JS-divergence for each histogram with $n = 30$ in *NYC*. **c** Median JS-divergence for varying K in *TKY*. **d** JS-divergence for each histogram with $n = 40$ in *TKY*

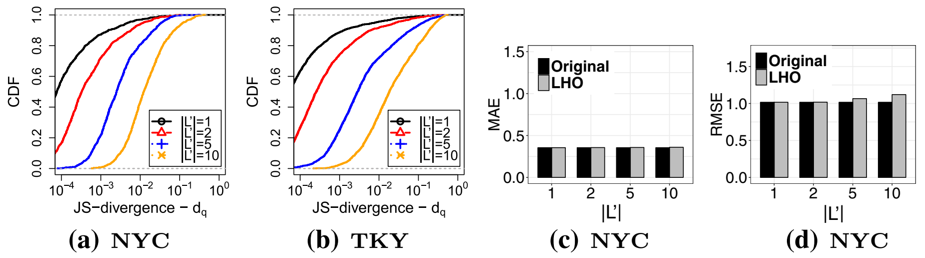


Fig. 10 Cumulative Distribution Function of JS-divergence (i.e., ratio of histograms with JS-divergence at most equal to a score in x axis) versus $|L'|$ for: **a** *NYC*, and **b** *TKY*. Recommendation quality versus $|L'|$ w.r.t: **c** MAE, and **d** *RMSE*

into the bins of the nonsensitive locations, and (II) fewer bins to which the counts may be redistributed into, and, as demonstrated above, both (I) and (II) incur more distortion. However, the distributions of the original histograms are preserved well even when $|L'| = 10$, with 90% of them having a JS-divergence score of at most 0.07. The remaining histograms have higher scores because on average 70% of their locations are treated as sensitive.

Recommendation quality Figure 10c, d show that MAE and *RMSE* are not substantially affected by sanitization, for all tested $|L'|$ values. The change in MAE and *RMSE* is on average 0.1% and 2.4%, respectively. This suggests that recommendation quality is preserved fairly well.

5.2.2 Runtime performance for the *LHO* algorithm

We evaluate the runtime performance of *LHO* as a function of (I) n, (II) K, and (III) $|L'|$. To isolate the effect of each parameter, we vary just one and keep the other two fixed. We then examine the joint impact of all three parameters, which is given by the time complexity formula $O\left((n - |L'|) \cdot K^2 \cdot \log((n - |L'|) \cdot K^2)\right)$, because we used Dijkstra's algorithm with binary heap to find shortest paths (see Sect. 4.1). For brevity, we use λ to denote $(n - |L'|) \cdot K^2 \cdot \log((n - |L'|) \cdot K^2)$. Thus, we expect the runtime to be linear in λ.

Impact of histogram length n We show that runtime increases with n, in Fig. 11a, b. This is because, when n is larger, there are more bins into which the counts may be redistributed. More bins means that the multipartite graph G_{TR}, created by *LHO*, has more layers (and consequently more nodes and edges).

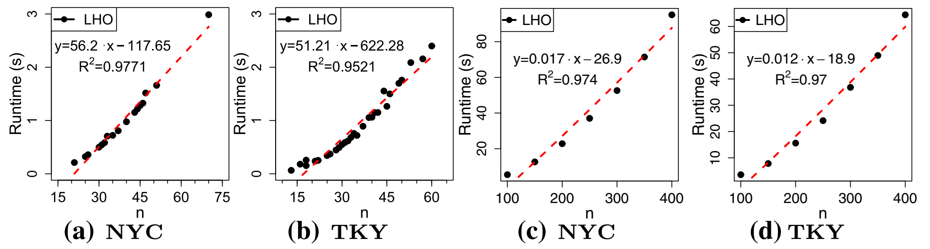


Fig. 11 Runtime versus length n, for each histogram with $K = 20$ in: **a** *NYC*, and **b** *TKY*. Runtime versus length n, for synthetic histograms with varying n, $K = 20$, and: **c** $N = 192$, **d** $N = 642$

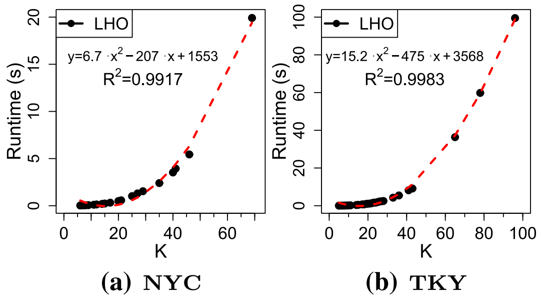


Fig. 12 Runtime versus K, for each histogram with: **a** $n = 30$, and **b** $n = 40$

Note also that runtime increases linearly with n (i.e., the linear regression models in Fig. 11a, b are good fit), as expected by the time complexity analysis (see Sect. 4.1), and that the algorithm took less than 3 seconds. We also show that runtime increases linearly with n when the algorithm is applied to the synthetic histograms, which are more demanding to sanitize (see Fig. 11c, d).

Impact of total frequency of sensitive locations K We show that runtime increases with K, in Fig. 12a, b. This is because there are more counts that are redistributed into the bins of nonsensitive locations when K is larger. That is, the graph G_{TR} contains more edges and nodes. The runtime increases approximately quadratically with K (i.e., the quadratic regression models in Fig. 12a, b are good fit), as expected by the time complexity analysis (see Sect. 4.1), and *LHO* took less than 100 seconds.

Impact of number of sensitive locations $|L'|$ We show that runtime increases with $|L'|$, in Fig. 13a, b. This is because there are (I) more counts that need to be redistributed into the bins of the nonsensitive locations, and (II) fewer bins to which the counts may be redistributed to, and, as demonstrated above, the impact of more counts on runtime is larger than that of fewer bins (quadratic increase vs. linear decrease). For example, 95% of the histograms in the *NYC* dataset take less than 1 second to be sanitized when $|L'| = 1$, but the corresponding percentage was 25% when $|L'| = 10$. However, the algorithm remains efficient even for $|L'| = 10$, with 99% of the histograms in *NYC* requiring less than 5 minutes to be sanitized.

Joint impact of $n, K, |L'|$ In Fig. 13c, d, we report results for all histograms in NYC and TKY, respectively. Note that runtime increases linearly with $\lambda = (n - |L'|) \cdot K^2 \cdot \log((n - |L'|) \cdot K^2)$

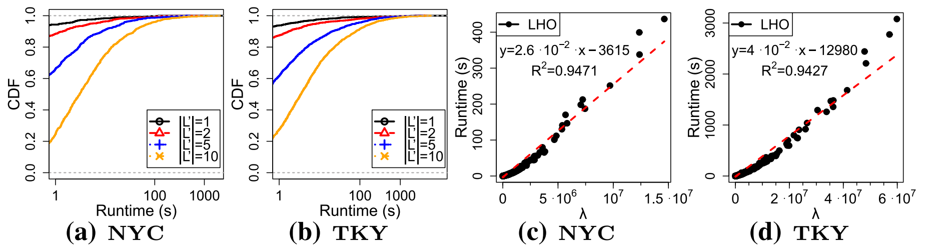


Fig. 13 Cumulative Distribution Function of runtime (i.e., ratio of histograms with runtime at most equal to a score in x axis) versus $|L'|$ for: **a** each histogram in *NYC*, and **b** each histogram in *TKY*. Runtime versus $\lambda = (n - |L'|) \cdot K^2 \cdot \log((n - |L'|) \cdot K^2)$ (i.e., joint impact of n, K, $|L'|$ according to the time complexity analysis) for: **c** each histogram in *NYC*, and **d** each histogram in *TKY*

(i.e., the linear regression models are good fit). This is in line with the time complexity analysis (see Sect. 4.1).

5.3 Target Resemblance

We evaluate the quality and runtime of *RO* and *RH*, as a function of (I) n, (II) N, and (III) ϵ. We additionally examine the impact of the target histogram H'' on the runtime, as well as the runtime of *RO* and *RH* when applied to histograms with large (up to the maximum possible) length. To measure quality, we use JS-divergence (see [33] for similar results w.r.t. L_2 distance). Unless stated otherwise, H'' is a "uniform" histogram that has the same size, N, and length, n, as H, and each of its counts is approximately equal to $\frac{N}{n}$. Aiming to resemble a uniform histogram indicates a user with strong privacy requirements, since the uniform distribution has the maximum entropy (i.e., provides the least information about the frequencies in H to an attacker with no knowledge except N and n). Moreover, uniform target histograms are difficult to resemble, because the original histograms typically follow skewed distributions.

Impact of length n To illustrate the impact of n on quality and privacy, we present results obtained for randomly selected histograms of varying n and $N = 100$. We do not report the median of all histograms of certain n, because the results followed skewed distributions.

5.3.1 Quality and privacy for the *RO* algorithm and the *RH* heuristic

We show that the privacy measure d_p (JS-divergence) decreases with n, in Fig. 14a, b. This is because the larger number of bins gives more choices to the methods to reduce d_p without substantially increasing d_q. Note, *RO* and *RH* achieve very similar results, suggesting that *RH* is effective: the d_p values for *RH* were no more than 1% and 2.1% higher for *NYC* and *TKY*, respectively.

We also show that the quality measure d_q (JS-divergence) is not affected by n and, as expected, it does not exceed the threshold ϵ, in Fig. 14c, d. *RO* finds solutions with larger d_q than *RH*. This is because *RH* works in a greedy fashion. That is, the initial bins are sanitized heavily, which increases d_q and does not leave much room for sanitizing the subsequent bins without exceeding ϵ.

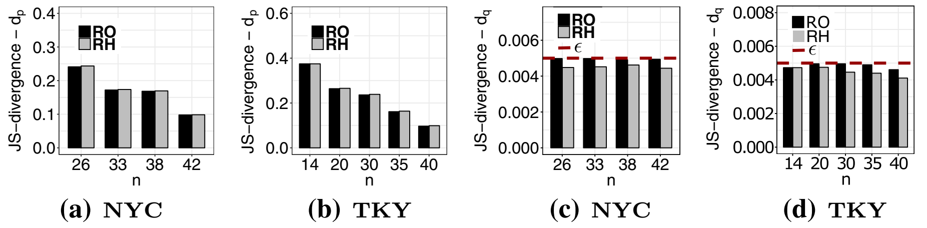
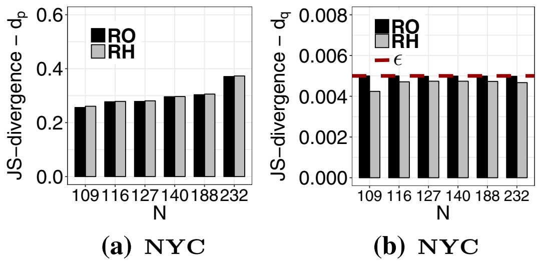


Fig. 14 d_p versus length n for histograms with $N = 100$ in: **a** *NYC*. **b** *TKY*. d_q versus length n for histograms with $N = 100$ in: **c** *NYC*. **d** *TKY*

Fig. 15 **a** d_p versus size N, **b** d_q versus size N, for histograms with $n = 25$ in *NYC*



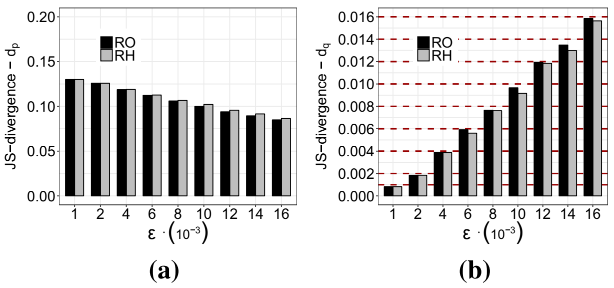


Fig. 16 **a** d_p versus threshold ϵ, **b** d_q versus threshold ϵ, for a histogram with $n = 40$ and $N = 100$ in *NYC*

Impact of size N To illustrate the impact of N on quality and privacy, we present results obtained for randomly selected histograms of varying N with $n = 25$ for *NYC*. The results for *TKY* are similar (omitted). We do not report the median of all histograms of certain N, because the results followed skewed distributions.

We show that the privacy measure d_p (JS-divergence) increases with N, in Fig. 15a. This is because there are more counts that need to change (increase or decrease) to minimize d_p subject to $d_q \leq \epsilon$. The results for *RH* are very close to those for *RO*; the d_p scores for *RH* are no more than 1.8% larger. This suggests that *RH* is an effective heuristic.

We also show that the quality measure d_q (JS-divergence) is not affected by N and that it does not exceed the threshold ϵ, in Fig. 15b. Again, *RO* finds solutions with larger d_q than *RH*. This is because, due to its greedy nature, *RH* sanitizes heavily the first bins, which increases d_q and prevents the sanitization of subsequent bins without exceeding ϵ.

Impact of threshold ϵ We show that d_p (JS-divergence) decreases with ϵ, in Fig. 16a. This is because both *RO* and *RH* consider a larger space of possible solutions when ϵ is larger, and thus they are able to find a better solution with respect to d_p.

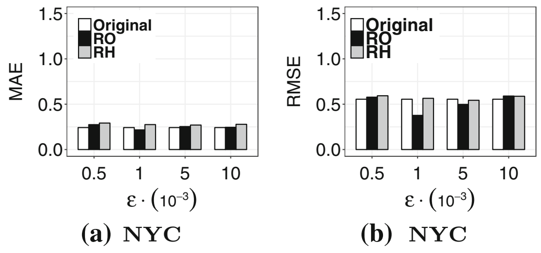


Fig. 17 Recommendation quality for varying ϵ w.r.t.: **a** MAE, and **b** $RMSE$

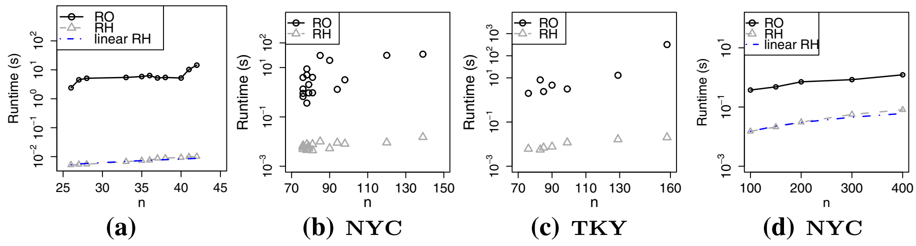


Fig. 18 Runtime versus length n, for: **a** each histogram with $N = 100$ in *NYC*, **b** each histogram with $n > 75$ in *NYC*, **c** each histogram with $n > 75$ in *TKY*, and **d** synthetic histograms with varying length and $N = 192$

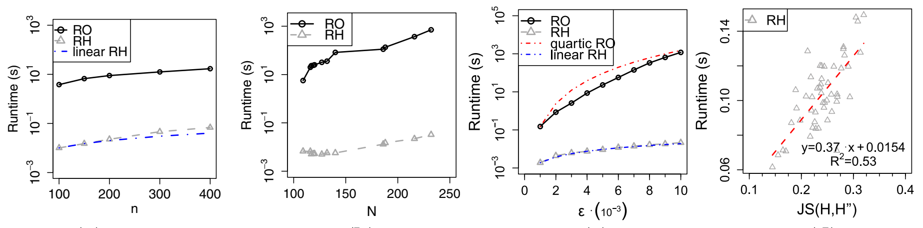
In addition, the results for *RH* and *RO* are very similar; the d_p for *RH* is at most 2.4% (on average 0.5%) higher than that for *RO*. We also show that the quality measure d_q (JS-divergence) for both *RO* and *RH* is close to ϵ, in Fig. 16b. Again, the d_q scores of *RO* are slightly larger than those of *RH*, because *RH* works in a greedy fashion, as explained above.

Recommendation quality Figure 17a, b shows that MAE and *RMSE* are not substantially affected by sanitization, for all tested ϵ values. This suggests that recommendation quality is preserved well. In some cases, the MAE and *RMSE* scores for the sanitized histograms were lower (better) than those for the original ones. This is because the recommendation scores for these histograms approach their corresponding true location counts after sanitization.

5.3.2 Runtime performance for the *RO* algorithm and the *RH* heuristic

Impact of length n We show that the runtime of both *RO* and *RH* increases with n, in Fig. 18a. This is because *RO* runs on a multipartite graph, G_{TR}, with more layers n, and *RH* needs to consider more bins n. *RH* is at least two orders of magnitude more efficient than *RO*. Note that *RH* scales close to linearly with n, which shows that the efficiency of *RH* is better than what is predicted by the worst-case time complexity analysis in Sect. 4.3.

To further investigate the impact of length on runtime, we apply *RO* and *RH* to each histogram with length larger than 75 in *NYC* and *TKY* (see Fig. 18b, c). There are 17 and 7 such histograms in *NYC* and *TKY*, respectively. These histograms are generally demanding to sanitize, because they also have large size (up to 2061). Again, we observe that *RH* is more efficient than *RO* by at least two orders of magnitude. In these experiments, we use $\epsilon = 10^{-5}$. For larger ϵ values, the difference between the two algorithms increases, because *RH* scales better than *RO* with respect to ϵ, as explained above. Repeating the same experiment using the synthetic histograms (see Figs. 18d, 19a), we find that both *RO* and *RH* scale well with n, and *RH* scales close to linearly with n.



(a) TKY **(b)** **(c)** **(d)**

Fig. 19 Runtime versus: **a** Length n, for synthetic histograms with varying length and $N = 642$. **b** Size N, for each histogram with $n = 25$ in *NYC*. **c** Threshold ϵ, for a histogram with $n = 40$ and $N = 100$ in *NYC*. **d** JS-divergence between a histogram H with $n = 40$ and $N = 100$ in *NYC* and different target histograms with increasing JS-divergence from H

Impact of size N We show that the runtime of both *RO* and *RH* increases with N, in Fig. 19b. This is because the multipartite graph G_{TR} built by *RO* has more nodes $O(N \cdot n)$ and thus more paths, and *RH* needs to consider more "moves" from source to destination bins. Again, *RH* is at least two orders of magnitude more efficient than *RO*.

Impact of threshold ϵ We show that the runtime of both *RO* and *RH* increases with ϵ, in Fig. 19c. This is because, when ϵ is larger, the multipartite graph built by *RO* has more edges and thus more paths, and *RH* considers more "moves" from source to destination bins. *RH* is at least two orders of magnitude more efficient than *RO*, and it scales better with ϵ, i.e., linearly versus quadratically (proportionally to ϵ^4). This suggests that *RH* is a practical heuristic for large ϵ values, given that it produces solutions similar to those of *RO*.

Impact of target histogram H'' We show that the runtime of *RH* increases with the distance $JS(H, H'')$, for different target histograms H'', in Fig. 19d. This is because *RH* has more choices (i.e., there are more ways to transfer the counts of a source bin to a destination bin, when $JS(H, H'')$ is larger). In this experiment, we use $\epsilon = 0.5$, because the runtimes with the default ϵ value are too small (few milliseconds) to obtain a meaningful result. We do not report the result for *RO*, because its runtime is not affected by the target histograms. The reason is that *RO* builds the same multipartite graph for each target histogram H'', since H'' has the same length n and size N with H.

6 Related work

This paper is at the intersection of location privacy and histogram privacy, which are discussed in Sects. 6.1 and 6.2, respectively. We also discuss privacy-preserving recommendation in Sect. 6.3, as a potential application of our methods.

6.1 Location privacy

Research on location privacy focuses on (I) location-based services (LBS), or (II) location data publishing.

Research on LBS is mostly inspired from applications running on GPS-enabled mobile devices like smartphones and tablets—but also cars. Consequently, it addresses privacy for users who need to send data on the fly (as they move about), to a server that will provide them with some useful service (e.g., the location of the nearest restaurant). Privacy mechanisms in such scenarios need to make protection decisions on the fly, without knowing the future locations that the user will visit [8,18,50,51]. For example, [50] proposes a method for preventing the inference of locations that have been or will be visited by a user, based on what the user shares at any moment with a location-based service. Other recent research protects sensitive spatiotemporal location sequences [1]. As another example, [18] proposes a method that prevents an LBS server from aggregating the locations sent by a user into a histogram and then associating this histogram with the user. The method perturbs the user's locations one by one, before they are sent to the LBS server, by adding noise to them in order to enforce the privacy notion of geo-indistinguishability [9].

Research on location data publishing is inspired from the publication of large datasets, possibly as a database. Consequently, it addresses more static scenarios, in which the whole dataset to be protected is given to the protection algorithm as input [4,10–12,15,43,53,54]. There are works showing the feasibility of attacks on *pseudonymized* data (i.e., data in which a user's identifying information is represented by a random id) [4], or on *completely anonymized* data (i.e., a sequence $(e_1, \ldots, e_n)$, where the event $e_i = (l, t)$, $i \in [1, n]$, represents a visit to location l at time t and is not associated to a specific user) [54]. For example, reference [54] shows how an attacker can use completely anonymized data to associate a user with their event subsequence (*path*). There are also works [11,12,15,43,53] which propose methods for anonymizing user-specific location data (i.e., a dataset where each record corresponds to a different user and contains a sequence of locations visited by the user and/or the time that these visits occurred). For example, reference [53] proposes algorithms for preventing the inference of a user's sensitive locations by an attacker knowing a subsequence of the user's locations. The algorithms of [53] use *suppression* (deletion) of locations and *splitting* of user sequences into carefully selected subsequences.

Yet, no research in location privacy has aimed to protect histograms of locations. The object/fact to be protected has been either a single location (in the LBS setting), or a (sub)sequence of locations (in the location data publishing setting). However, protecting single locations separately provides no guarantee about the effect on the histogram as a whole. It could happen that, e.g., each individual location is replaced with another location, so no single location is disclosed/compromised, but the histogram as a whole is very similar or even identical with the original one. Similarly, protecting location data could again lead to the same problem. It could happen that individual locations in a user's sequence are modified, but the histogram remains unprotected. Thus, works on LBS or location data publishing cannot be used as alternatives to our approach.

6.2 Histogram privacy

Research on histogram privacy is inspired from applications where a histogram is published as a statistical summary (approximation) of the distribution of an attribute in a (relational) dataset. For example, consider a dataset, where each record contains the zip-code of a different individual. The distribution of the zip-code attribute in the dataset can be represented with a histogram, where each bin is associated with a different zip-code value and the bin frequency (count) is the number of individuals in the dataset who live in the zip-code. Publishing such histograms is useful for performing count query answering and data mining tasks (e.g., clustering), but it may lead to the disclosure of sensitive information about individuals [2, 16,23,26,45,61,68]. For instance, consider an adversary who knows the names of all three individuals, i_1, i_2, and i_3, in a (non-released) dataset but the zip-codes of only i_1 and i_2. When the published histogram contains the count of each zip-code in the dataset, the adversary can infer the zip-code of i_3 from the histogram. To prevent this type of disclosure, the frequencies in the histogram are perturbed, typically by noise addition, in order to satisfy *differential privacy* [17]. Informally, differential privacy ensures that the inferences that can be made by an adversary about an individual will be approximately independent of whether the individual's record is included in the dataset or not.

Several works have applied differential privacy to sanitize histograms [2,16,23,26,45, 61,68]. A straightforward way to achieve this is by adding noise to the frequency of each bin of the histogram, according to the Laplace mechanism [17]. However, this procedure results in excessive utility loss [2]. Therefore, existing works [2,16,23,26,45,61,68] employ clustering to reduce the loss of utility, in three steps: (I) they cluster bins with similar frequencies together. (II) They apply the Laplace mechanism to the average (mean or median) of the frequencies in each cluster, to obtain a "noisy center" of the cluster. (III) They publish a histogram where each frequency bin in each cluster is replaced by the noisy center of its corresponding cluster. While clustering incurs some utility loss, it reduces the noise that is added by the Laplace mechanism, leading to better overall utility. Specifically, the works of [2,61] require each cluster to be formed of adjacent bins, while the work of [23] requires each cluster to have the same number of bins. Subsequent works [16,26,45,68] lift these restrictions to further improve utility. For example, reference [68] proposes a clustering framework, which can be instantiated by optimal or heuristic algorithms that trade-off the utility loss incurred by clustering with the utility loss incurred by the Laplace mechanism.

At a high level, our work is similar to the works in [2,16,23,26,45,61,68], in that it aims to protect a histogram (or it can be applied to each histogram in a dataset of histograms). However, it differs from the works in [2,16,23,26,45,61,68] along two dimensions: (I) it considers a histogram that represents the locations associated to a *single user*, instead of a histogram representing the values of many different individuals in an attribute of an underlying dataset. (II) It sanitizes a histogram by redistributing counts between bins, as specified by Problems 3.1, 3.2, and 3.3, instead of adding noise into the counts. Thus, the methods in [2,16,23,26,45,61,68] cannot be used to deal with the problems we consider. In fact, applying any of the methods in [2,16,23,26,45,61,68] to a histogram that represents the locations of a single user would simply prevent the inference of the exact frequencies (counts) of locations in the user's histogram. It would not protect against the disclosure of visits to sensitive locations (i.e., it cannot solve the *SLH* problem), nor against the disclosure of the fact that the histogram is similar/dissimilar to a target histogram (i.e., it cannot solve the *TA*/*TR* problem).

A different, less related class of works can be used to protect a histogram by making it indistinguishable within a set of histograms that is published [20,60]. These works differ from ours in their setting, in their privacy notion, or both. They differ in terms of setting because they consider a set of histograms (or more generally, vectors of frequencies [20,60]) rather than a single histogram with the location information of a single user. They differ in terms of privacy notion because they aim to prevent the disclosure of the identity of individuals, from the published set of histograms (i.e., the association of a histogram with identity information that is known to an attacker), rather than the inference of location information from a single histogram.

6.3 Privacy-preserving recommendation

There are several privacy-preserving recommendation methods. Most of them (e.g., [34,46]) assume there is a trusted server that applies privacy protection (e.g., anonymization) jointly to the data of many users. Unlike these methods, we assume a different setting, in which the user protects their histogram by themselves. Our setting is conceptually similar to the untrusted server setting [42,48,49], in which a user protects their data prior to disseminating them. Specifically, [48,49] propose methods in which a user applies differential privacy, while [42] proposes a method in which the user applies *randomized perturbation*. The privacy objective of these methods is to prevent the inference of exact user values. In contrast, we do not directly aim to prevent the inference of exact user values: our privacy notions are formalized by the *SLH* and *TA*/*TR* problems. Also, we do not require that the protected histograms will be used in the task of recommendation, although we experimentally show that the protected histograms that are produced by our approach allow preserving the accuracy of recommendation fairly well.

7 Conclusion

In this paper, we propose two new notions of histogram privacy, Sensitive Location Hiding and Target Avoidance/Resemblance, which lead to the following optimization problems: the Sensitive Location Hiding problem (*SLH*), which seeks to enforce the notion of Sensitive Location Hiding with optimal quality, and the Target Avoidance/Resemblance (*TA*/*TR*) problem, which seeks to enforce Target Avoidance/Resemblance with bounded quality loss. We also propose optimal algorithms for each problem, as well as an efficient heuristic for the *TA*/*TR* problem. Our experiments demonstrate that our methods are effective at preserving the distribution of locations in a histogram, as well as the quality of recommendations based on these locations, while being fairly efficient.

A Appendix

A.1 Proof of weak NP-hardness for the *SLH* problem

We first reduce the weakly NP-hard *Multiple Choice Knapsack* (MCK) problem [24] to the special case of *SLH*, where $|L'| = 1$. The MCK problem is defined as follows[4]:

$$\min \sum_{i \in [1,m]} \sum_{j \in C_i} c_{ij} \cdot x_{ij} \tag{A.1}$$

[4] The problem also appears with $\geq$ in constraint I [52]. This variation is referred to as $MCK_{\geq}$ and can be transformed to MCK in polynomial time [24].

subject to: (I) $\sum_{i\in[1,m]}\sum_{j\in C_i} w_{ij}\cdot x_{ij} = b$, (II) $\sum_{j\in C_i} x_{ij} = 1,\ \ i = 1,\dots m$, and (III) $x_{ij}\in\{0,1\},\ \ i = 1,\dots,m,\ \ \ j\in C_i$.

In MCK, we are given a set of elements subdivided into m, mutually exclusive classes, $C_1,\dots,C_m$, and a knapsack. Each class C_i has $|C_i|$ elements. Each element $j\in C_i$ has a cost $c_{ij}\geq 0$ and a weight w_{ij}. The goal is to minimize the total cost (Eq. A.1) by filling the knapsack with one element from each class (constraint II), such that the weights of the elements in the knapsack satisfy the constraint I, where $b\geq 0$ is a constant. The variable x_{ij} takes a value 1, if the element j is chosen from class C_i and 0 otherwise (constraint III).

We map a given instance $\mathcal{I}_{MCK}$ to an instance $\mathcal{I}_{SLH}$ of the special case of *SLH* in polynomial time, as follows:

(I) Each class C_i, $i\in[1,m]$, is mapped to a location $L_i\notin L'$ whose count $f(L_i)$ in H is arbitrary.
(II) A sensitive location $L_{m+1}\in L'$ (without loss of generality) is considered. The count of L_{m+1} in H is set to $f(L_{m+1}) = b$. Thus, $H = (f(L_1),\dots,f(L_m),b)$.
(III) Each element x_{ij} with weight w_{ij} and cost c_{ij} is mapped to an operation on H, which decreases $f(L_{m+1})$ by w_{ij} and increases $f(L_i)$ by w_{ij} (i.e., transfers w_{ij} visits from L_{m+1} to L_i) and incurs $q(H,H'[i]) = c_{ij}$. If there are multiple operations such that $q(H,H'[i]) = c_{ij}$ (e.g., when q is the L_1 distance), we select one arbitrarily. When $x_{ij} = 1$, its corresponding operation is applied to H. The result of applying all operations on H is referred to as the sanitized histogram H'.

We prove the correspondence between a solution $\mathcal{S}$ to $\mathcal{I}_{MCK}$ and a solution H' to $\mathcal{I}_{SLH}$, as follows: we first prove that, if $\mathcal{S}$ is a solution to $\mathcal{I}_{MCK}$, then H' is a solution to $\mathcal{I}_{SLH}$. Since $\sum_{i\in[1,m]}\sum_{j\in C_i} w_{ij}\cdot x_{ij} = b$, $f(L_{m+1})$ is decreased by b. Thus, $H'[m+1] = 0$ (i.e., all visits to L_{m+1} are transferred to nonsensitive locations) and $\sum_{i\in[1,m+1]} H'[i] = \sum_{i:L_i\notin L'} H'[i] = |H|_1$ (i.e., H' has the same size with H). By construction, H' has also the same length with H. Since $\sum_{i\in[1,m]}\sum_{j\in C_i} c_{ij}\cdot x_{ij}$ is minimum, $\sum_{i\in[1,m]} q(H,H'[i])$ is minimum. Also, $q(H,H'[m+1])$ (i.e., the loss for transferring all visits from L_{m+1} to nonsensitive locations) is constant. Thus, $d_q(H,H') = \sum_{i\in[1,m+1]} q(H,H'[i])$ is minimum. Therefore, H' is a solution to $\mathcal{I}_{SLH}$.

We now prove that, if H' is a solution to $\mathcal{I}_{SLH}$, then $\mathcal{S}$ is a solution to $\mathcal{I}_{MCK}$. Since $\sum_{i\ s.t.\ L_i\notin L'} H'[i] = |H|_1$, it holds that $H'[m+1] = 0$. Thus, $f(L_{m+1})$ is decreased by b (all visits to L_{m+1} were transferred to nonsensitive locations) and $\sum_{i\in[1,m]}\sum_{j\in C_i} w_{ij}\cdot x_{ij} = b$. Since $d_q(H,H') = (\sum_{i\in[1,m]} q(H,H'[i])) + q(H,H'[m+1])$ is minimum and $q(H,H'[m+1])$ is constant, $\sum_{i\in[1,m]} q(H,H'[i])$ is minimum. This implies that $\sum_{i\in[1,m]}\sum_{j\in C_i} c_{ij}\cdot x_{ij}$ is minimum. Thus, $\mathcal{S}$ is a solution to $\mathcal{I}_{MCK}$.

Therefore, the special case of the *SLH* problem with $|L'| = 1$ is weakly NP-hard, and, clearly, the *SLH* problem with $|L'|\geq 1$, is also weakly NP-hard.

A.2 Proof of weak NP-hardness for the *TR* problem

We reduce the weakly NP-hard *Multiple Choice Knapsack* ($MCK_{\geq}$) problem [24,52] to the *TR* problem. The $MCK_{\geq}$ problem is defined as follows:

$$\min \sum_{i\in[1,n]}\sum_{j\in C_i} c_{ij}\cdot x_{ij} \tag{A.2}$$

subject to: (I) $\sum_{i\in[1,n]}\sum_{j\in C_i} w_{ij}\cdot x_{ij}\geq b$, (II) $\sum_{j\in C_i} x_{ij} = 1, i = 1,\dots n$, and (III) $x_{ij}\in\{0,1\}, i = 1,\dots,n,\ j\in C_i$. In $MCK_{\geq}$, we are given a set of elements subdivided into

n, mutually exclusive classes, $C_1, \ldots, C_n$, and a knapsack. Each class C_i has $|C_i|$ elements. Each element $j \in C_i$ has a cost $c_{ij} \geq 0$ and a weight w_{ij}. The goal is to minimize the total cost (Eq. A.2) by filling the knapsack with one element from each class (constraint II), such that the weights of the elements in the knapsack satisfy the constraint I, where $b \geq 0$ is a constant. The variable x_{ij} takes a value 1, if the element j is chosen from class C_i and 0 otherwise (constraint III).

We map a given instance $\mathcal{I}_{MCK_{\geq}}$ to an instance $\mathcal{I}_{TR}$ of *TR* in polynomial time, as follows:

(I) Each class C_i, $i \in [1, n]$, is mapped to a location L_i, which has an arbitrary count in H and a possibly different, arbitrary count in H''.

(II) The constant ϵ is set to $n - \frac{b}{\max_{i \in [1,n], j \in C_i} w_{ij}}$.

(III) We choose $q()$ and $p()$ to be normalized in $[0, 1]$ such that each element x_{ij} with weight w_{ij} and cost c_{ij} is mapped to a value k_{ij} such that the following conditions hold: $q(H, H[i] + k_{ij}) = 1 - \frac{w_{ij}}{\max_{i \in [1,n], j \in C_i} w_{ij}}$ and $p(H[i] + k_{ij}, H'') = \frac{c_{ij}}{\max_{i \in [1,n], j \in C_i} c_{ij}}$. The normalization of $q()$ and $p()$ can be done in polynomial time because $q()$ and $p()$ can take $O(N \cdot n)$ values. If there are multiple values of k_{ij} satisfying the two conditions, one of these values is selected arbitrarily. When $x_{ij} = 1$, k_{ij} is added into $H[i]$, obtaining $H'[i] = H[i] + k_{ij}$.

We prove the correspondence between a solution $\mathcal{S}$ to $\mathcal{I}_{MCK_{\geq}}$ and a solution H' to $\mathcal{I}_{TR}$:

We first prove that, if $\mathcal{S}$ is a solution to $\mathcal{I}_{MCK_{\geq}}$, then H' is a solution to $\mathcal{I}_{TR}$. Since $\sum_{i \in [1,n]} \sum_{j \in C_i} c_{ij} \cdot x_{ij}$ is minimum, $\sum_{i \in [1,n]} \big((\max_{i \in [1,n], j \in C_i} c_{ij}) \cdot p(H'[i], H'')\big)$ is minimum. Thus, $d_p(H', H'') = \sum_{i \in [1,n]} p(H'[i], H'')$ is minimum. Since $\sum_{i \in [1,n]} \sum_{j \in C_i} w_{ij} \cdot x_{ij} \geq b$ and $b = \max_{i \in [1,n], j \in C_i} w_{ij} \cdot (n - \epsilon)$, it holds that $\sum_{i \in [1,n]} (1 - q(H, H[i] + k_{ij})) \geq n - \epsilon$. This implies $n - d_q(H, H') \geq n - \epsilon$ and $d_q(H, H') \leq \epsilon$. Therefore, H' is a solution to $\mathcal{I}_{TR}$. We now prove that, if H' is a solution to $\mathcal{I}_{TR}$, then $\mathcal{S}$ is a solution to $\mathcal{I}_{MCK_{\geq}}$. Since $d_p(H', H'') = \sum_{i \in [1,n]} p(H'[i], H'')$ is minimum, $\sum_{i \in [1,n]} \big((\max_{i \in [1,n], j \in C_i} c_{ij}) \cdot p(H'[i], H''[i])\big)$ is minimum. This implies that $\sum_{i \in [1,n]} \sum_{j \in C_i} c_{ij} \cdot x_{ij}$ is minimum. Since $d_q(H, H') = \sum_{i \in [1,n]} q(H, H'[i]) \leq \epsilon$ and $\epsilon = n - \frac{b}{\max_{i \in [1,n], j \in C_i} w_{ij}}$, it holds that $n - \sum_{i \in [1,n]} q(H, H'[i]) \geq n - \epsilon = \frac{b}{\max_{i \in [1,n], j \in C_i} w_{ij}}$. This implies $\sum_{i \in [1,n]} (1 - q(H, H[i] + k_{ij})) \geq \frac{b}{\max_{i \in [1,n], j \in C_i} w_{ij}}$ and $\sum_{i \in [1,n]} \sum_{j \in C_i} w_{ij} \cdot x_{ij} \geq b$. Thus, $\mathcal{S}$ is a solution to $\mathcal{I}_{MCK_{\geq}}$. Therefore, *TR* is weakly NP-hard.

A.3 Reduction from the *TA* to the *TR* problem

The *TA* problem can be reduced to *TR* in polynomial time: given an instance $\mathcal{I}_{TA}$ of *TA*, we can construct an instance $\mathcal{I}_{TR}$ of *TR* in polynomial time, by mapping H and H'' to histograms $H_{TR} = H$ and $H''_{TR} = H''$, defining $d_p(H'_{TR}, H''_{TR}) = \frac{1+1}{d_p(H', H'')+1} \in (0, 2]$ and $d_q(H_{TR}, H'_{TR}) = d_q(H, H')$, and setting $\epsilon_{TR} = \epsilon$. Given a feasible solution H'_{TR} of $\mathcal{I}_{TR}$, we can map it back to a feasible solution H' of $\mathcal{I}_{TA}$ with cost $d_p(H', H'') = \frac{1+1}{d_p(H'_{TR}, H''_{TR})} - 1 \geq 0$ in polynomial time. This requires constructing $H' = H'_{TR}$, which clearly is a solution to $\mathcal{I}_{TA}$.

References

1. Abul O, Bayrak C (2018) From location to location pattern privacy in location-based services. Knowl Inf Syst 56(3):533–557. https://doi.org/10.1007/s10115-017-1146-x

2. Acs G, Castelluccia C, Chen R (2012) Differentially private histogram publishing through lossy compression. In: ICDM, pp 1–10
3. Ağır B, Huguenin K, Hengartner U, Hubaux JP (2016) On the privacy implications of location semantics. Proc Priv Enhanc Technol 2016(4):165–183
4. Arapinis M, Mancini LI, Ritter E, Ryan M (2014) Privacy through pseudonymity in mobile telephony systems. In: NDSS
5. Bettini C, Riboni D (2015) Privacy protection in pervasive systems: state of the art and technical challenges. Pervasive Mobile Comput 17(Part B):159–174
6. Bóna M (2011) A walk through combinatorics: an introduction to enumeration and graph theory. World Scientific, Singapore
7. Breese JS, Heckerman D, Kadie C (1998) Empirical analysis of predictive algorithms for collaborative filtering. In: Uncertainty in artificial intelligence, pp 43–52
8. Chatzikokolakis K, Andrés ME, Bordenabe NE, Palamidessi C (2013) Broadening the scope of differential privacy using metrics. In: PETS, pp 82–102
9. Chatzikokolakis K, Palamidessi C, Stronati M (2015) Location privacy via geo-indistinguishability. ACM SIGLOG News 2(3):46–69
10. Chen R, Acs G, Castelluccia C (2012) Differentially private sequential data publication via variable-length n-grams. In: CCS, pp 638–649
11. Chen R, Fung BCM, Mohammed N, Desai BC, Wang K (2013) Privacy-preserving trajectory data publishing by local suppression. Inf Sci 231:83–97
12. Cicek AE, Nergiz ME, Saygin Y (2014) Ensuring location diversity in privacy-preserving spatio-temporal data publishing. VLDB J 23(4):609–625
13. Damiani ML, Bertino E, Silvestri C et al (2010) The probe framework for the personalized cloaking of private locations. Trans Data Priv 3(2):123–148
14. Dao TH, Jeong SR, Ahn H (2012) A novel recommendation model of location-based advertising. Expert Syst Appl 39(3):3731–3739
15. Domingo-Ferrer J, Trujillo-Rasua R (2012) Microaggregation- and permutation-based anonymization of movement data. Inf Sci 208:55–80
16. Doudalis S, Mehrotra S (2017) SORTAKI: framework to integrate sorting with differential private histogramming algorithms. In: PST
17. Dwork C, McSherry F, Nissim K, Smith A (2006) Calibrating noise to sensitivity in private data analysis. In: Theory of cryptography, pp 265–284
18. Fawaz K, Shin KG (2014) Location privacy protection for smartphone users. In: CCS. ACM, New York, pp 239–250
19. Gedik B, Liu L (2008) Protecting location privacy with personalized k-anonymity: architecture and algorithms. IEEE Trans Mob Comput 7(1):1–18
20. Ghinita G, Karras P, Kalnis P, Mamoulis N (2007) Fast data anonymization with low information loss. In: VLDB, pp 758–769
21. Ghose A, Li B, Liu S (2015) Digitizing offline shopping behavior towards mobile marketing. In: International conference on information systems
22. Hasan S, Ukkusuri SV (2015) Location contexts of user check-ins to model urban geo life-style patterns. PLoS ONE 10(5):e0124819
23. Kellaris G, Papadopoulos S (2013) Practical differential privacy via grouping and smoothing. In: PVLDB, pp 301–312
24. Kellerer H, Pferschy U, Pisinger D (2004) The multiple-choice knapsack problem. Springer, Berlin, pp 317–347
25. Le Boudec JY (2010) Performance evaluation of computer and communication systems. EPFL Press, Lausanne
26. Li H, Cui J, Lin X, Ma J (2016) Improving the utility in differential private histogram publishing: theoretical study and practice. In: IEEE big data, pp 1100–1109
27. Li T, Li N, Zhang J (2009) Modeling and integrating background knowledge in data anonymization. In: ICDE, pp 6–17
28. Lian D, Ge Y, Zhang F, Yuan NJ, Xie X, Zhou T, Rui Y (2015) Content-aware collaborative filtering for location recommendation based on human mobility data. In: ICDM, pp 261–270
29. Lian D, Zhao C, Xie X, Sun G, Chen E, Rui Y (2014) GeoMF: joint geographical modeling and matrix factorization for point-of-interest recommendation. In: KDD, pp 831–840
30. Lin J (1991) Divergence measures based on the Shannon entropy. IEEE Trans Inf Theor 37(1):145–151
31. Loukides G, Gkoulalas-Divanis A, Shao J (2013) Efficient and flexible anonymization of transaction data. Knowl Inf Syst 36(1):153–210
32. Loukides G, Gwadera R (2015) Optimal event sequence sanitization. In: SIAM SDM, pp 775–783

33. Loukides G, Theodorakopoulos G (2019) Location histogram privacy by sensitive location hiding and target histogram avoidance/resemblance (extended version). arXiv:1912.00055
34. McSherry F, Mironov I (2009) Differentially private recommender systems: building privacy into the Netflix prize contenders. In: KDD, pp 627–636
35. Melville P, Sindhwani V (2017) Recommender systems. Springer, Berlin, pp 1056–1066
36. Melville P, Yang SM, Saar-Tsechansky M, Mooney R (2005) Active learning for probability estimation using Jensen–Shannon divergence. In: ECML, pp 268–279
37. Mikians J, Gyarmati L, Erramilli V, Laoutaris N (2012) Detecting price and search discrimination on the internet. In: HotNets, pp 79–84
38. Mikians J, Gyarmati L, Erramilli V, Laoutaris N (2013) Crowd-assisted search for price discrimination in e-commerce: first results. In: CoNext, pp 1–6
39. Murakami T, Kanemura A, Hino H (2017) Group sparsity tensor factorization for re-identification of open mobility traces. IEEE Trans Inf Forensics Sec 12(3):689–704
40. Nielsen F, Nock R, Amari Si (2014) On clustering histograms with k-means by using mixed α-divergences. Entropy 16:3273–3301
41. Papadimitriou CH (1994) Computational complexity. Addison-Wesley, Boston
42. Polat H, Du W (2003) Privacy-preserving collaborative filtering using randomized perturbation techniques. In: ICDM, pp 625–628
43. Poulis G, Skiadopoulos S, Loukides G, Gkoulalas-Divanis A (2014) Apriori-based algorithms for k^m-anonymizing trajectory data. Trans Data Priv 7(2):165–194
44. Pugliese L, Di P, Guerriero F (2013) Dynamic programming approaches to solve the shortest path problem with forbidden paths. Optim Methods Softw 28(2):221–255
45. Qardaji W, Yang W, Li N (2013) Understanding hierarchical methods for differentially private histograms. PVLDB 6(14):1954–1965
46. Sakuma J, Osame T (2018) Recommendation with k-anonymized ratings. Trans Data Priv 11(1):47–60
47. Sedgewick R, Wanye K (2011) Algorithms. Addison-Wesley, Boston
48. Shen Y, Jin H (2014) Privacy-preserving personalized recommendation: an instance-based approach via differential privacy. In: ICDM, pp 540–549
49. Shen Y, Jin H (2016) Epicrec: towards practical differentially private framework for personalized recommendation. In: CCS, pp 180–191
50. Shokri R, Theodorakopoulos G, Troncoso C (2017) Privacy games along location traces: a game-theoretic framework for optimizing location privacy. ACM Trans Privacy Secur (TOPS) 19(4):11
51. Shokri R, Theodorakopoulos G, Troncoso C, Hubaux JP, Boudec JYL (2012) Protecting location privacy: optimal strategy against localization attacks. In: CCS, pp 617–627
52. Sinha P, Zoltners AA (1979) The multiple-choice knapsack problem. Oper Res 27(3):503–515
53. Terrovitis M, Poulis G, Mamoulis N, Skiadopoulos S (2017) Local suppression and splitting techniques for privacy preserving publication of trajectories. IEEE Trans Knowl Data Eng 29(7):1466–1479
54. Tsoukaneri G, Theodorakopoulos G, Leather H, Marina MK (2016) On the inference of user paths from anonymized mobility data. In: IEEE European symposium on security and privacy, pp 199–213
55. Velardi P, Cucchiarelli A, Petit M (2007) A taxonomy learning method and its application to characterize a scientific web community. IEEE Trans Knowl Data Eng 19(2):180–191
56. Vissers T, Nikiforakis N, Bielova N, Joosen W (2014) Crying wolf? On the price discrimination of online airline tickets. In: HotPETs 2014
57. Weichselbaumer D (2003) Sexual orientation discrimination in hiring. Labour Econ 10(6):629–642
58. Wu Z, Li G, Liu Q, Xu G, Chen E (2018) Covering the sensitive subjects to protect personal privacy in personalized recommendation. IEEE Trans Serv Comput 11(3):493–506
59. Xiao X, Tao Y (2006) Personalized privacy preservation. In: Proceedings of the 2006 ACM SIGMOD international conference on management of data, SIGMOD'06. ACM, New York, pp 229–240
60. Xu J, Wang W, Pei J, Wang X, Shi B, Fu AWC (2006) Utility-based anonymization using local recoding. In: KDD, pp 785–790
61. Xu J, Zhang Z, Xiao X, Yang Y, Yu G, Winslett M (2013) Differentially private histogram publication. VLDB J 22(6):797–822
62. Yang D (2017) https://sites.google.com/site/yangdingqi/home/foursquare-dataset
63. Yang D, Zhang D, Zheng VW, Yu Z (2015) Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. IEEE Trans Syst Man Cybern 45(1):129–142
64. Yao Z, Fu Y, Liu B, Liu Y, Xiong H (2016) POI recommendation: A temporal matching between POI popularity and user regularity. In: ICDM, pp 549–558
65. Yu Y, Chen X (2015) A survey of point-of-interest recommendation in location-based social networks. In: AAAI workshop on trajectory-based behavior analytics

66. Zaki MJ Jr, Meira W (2014) Data mining and analysis: fundamental concepts and algorithms. Cambridge University Press, Cambridge
67. Zhang H, Yan Z, Yang J, Tapia EM, Crandall DJ (2014) mFingerprint: privacy-preserving user modeling with multimodal mobile device footprints. In: SBP, pp 195–203
68. Zhang X, Chen R, Xu J, Meng X, Xie Y (2014) Towards accurate histogram publication under differential privacy. In: SIAM SDM, pp 587–595

Grigorios Loukides is currently an Assistant Professor (Lecturer) in the Department of Informatics, King's College London. Before joining King's, he was a Royal Academy of Engineering Research Fellow, hosted at Cardiff University, and prior to that he was a Postdoctoral Research Fellow at Vanderbilt University. Grigorios obtained a Ph.D. degree in Computer Science from Cardiff University. His research interests are in data privacy, data mining, and biomedical informatics.

George Theodorakopoulos is a Senior Lecturer at the School of Computer Science and Informatics, Cardiff University, which he joined in 2012. From 2007 to 2011, he was a Senior Researcher at the Ecole Polytechnique Federale de Lausanne (EPFL), Switzerland. He received the M.S. and Ph.D. degrees from the University of Maryland, College Park, MD, USA, in 2004 and 2007, in both electrical and computer engineering. His research interests are in data privacy and security in computer networks.