History Trajectory Privacy-preserving through Graph Partition

Zheng Huo
Information School
Renmin University of China
Beijing, China
huozheng@ruc.edu.cn

Yi Huang
Information School
Renmin University of China
Beijing, China
yeastyi@gmail.com

Xiaofeng Meng
Information School
Renmin University of China
Beijing, China
xfmeng.ruc@gmail.com

ABSTRACT
With rapid development of positioning techniques and location based services (LBS), locations and traces of moving objects are collected by service providers, the data will then be published for novel applications. Although analyzing and mining trajectories is useful for mobility-related applications, new challenges of trajectory privacy leakage arise accordingly. Trajectories contain rich spatio-temporal history information that may expose users' whereabouts and other personal privacy. At present, trajectory *k*-anonymity which aims at anonymizing *k* trajectories together on all sample points is one of the most popular techniques to protect trajectory privacy. The challenge lies in how to find trajectory *k*-anonymity sets. In this paper, a trajectory graph is constructed to simulate spatial relations of trajectories, based on which we propose to find trajectory *k*-anonymity sets through graph partition, which is proven *NP-complete*. We then propose a greedy partition method to find trajectory *k*-anonymity sets, as well as yielding low information loss. We run a series of experiments on both real-world and synthetic datasets, the results show the effectiveness of our method.

ACM Classification Keywords
H.2.m [Database Management]: Miscellaneous performance measures

General Terms
Algorithm

Author Keywords
Privacy-preserving, Data publication, Trajectory *k*-anonymity

INTRODUCTION

Recent years, location based services and positioning techniques developed fast based on the pervasive of location-aware devices, such as, GPS-enabled cell phones and PDAs, location sensors and RFID cards. Thus, location privacy of users may be exposed when requiring for services. Location privacy-preserving techniques has gained much attention in last a few years [6,7,9,12]. However, another privacy threaten arises with people's trajectories left behind and collected by service providers. The collected trajectories are then published for novel applications [3, 14, 17]. For example, analyzing trajectories of passengers in a certain area may help bussiness men making commercial decisions, such as where to build a restaurant or a shopping mall. Another example can be seen in traffic controlling systems, analyzing trajectories of vehicles in a city may help government to optimize traffic management strategy. Although publishing and analyzing trajectories is beneficial for mobility-related decision making processes, it may represent serious threats to users' privacy, since locations and trajectories contain rich spatio-temporal information, which may reveal ones' behavior patterns, living habits, health conditions, social customs etc.

When an adversary associates some sensitive information with a moving object's identity, a privacy violation happens. It is generally accepted that, simply removing identifiers may not prevent leakage of privacy, since adversaries may re-identify a victim through background information linkage, such as, by linkage of a trajectory with one's home and work place may identify a user from the published data. Suppose there are two neighbors Bob and Alice, Bob bears malice to Alice, he wants to know private information about Alice. Since they are neighbors, Bob knows where Alice lives, through their causal conversations, Bob knows where Alice works. The home address and work place can be regarded as background information. Through linkage of background information to published trajectory data, Bob tries everything to find from the published data which trajectory belongs to Alice and where Alice has visited, such as bars, clinics or other sensitive places during her way to work or back home. If Bob succeeds in associating a trajectory or sensitive locations of a trajectory with Alice, her privacy exposes through her everyday life trajectories.

In order to protect trajectory privacy, researchers have proposed several techniques, such as dummy trajectories confusion, suppression-based method and trajectory *k*-anonymity. Among these techniques, trajectory *k*-anonymity which tries to anonymize *k* trajectories together while balances well between privacy level and data utility is one of the most popular techniques in trajectory privacy-preserving community.

MLBS'11, September 18, 2011, Beijing, China.

However, how to find trajectories k-anonymity sets with minimum information loss is the key issue. Most existing methods try to find trajectory k-anonymity sets through trajectory clustering, however, these methods do not in general include much provision for satisfying constraints on the sizes of the anonymity sets, nor do they provide for systematic assignment of trajectories that do not obviously belong to any clusters. Moreover, clustering methods need to compute distances repeatedly to get better clusters, this may cause extra computation cost. So how to find trajectory k-anonymity sets is still a challenging problem. In this paper, we make the following contributions:

- We formalize trajectory k-anonymity to a graph partition problem based on the concept of trajectory graph we define. A trajectory graph is a weighted undirected graph which models spatial relations of trajectories.
- We define k-node partition which is proven *NP-complete* to achieve trajectory k-anonymity. A greedy k-node partition method is proposed to get an approximate result.
- We run a set of evaluations on both real-world and synthetic datasets. The results show advantages of our method.

The rest of the paper is organized as follows. Section 2 summarizes related works. Section 3 defines concepts and attack models we study in this paper. In section 4, we present our proposed solutions. Experiment evaluations are shown in section 5. Finally, section 6 concludes the paper.

RELATED WORKS

Trajectory data privacy is a rather young research area that has received a lot of interests in recent years. Relational data privacy protection methods are extended spatio-temporally to the community of trajectory data privacy. Main techniques in this community are in three types: dummy trajectory confusion, suppression-based method and trajectory k-anonymity.

Dummy trajectory confusion. Protecting trajectory privacy in a data publication perspective is first proposed in [16] with a simple dummy trajectories confusion method, the authors propose to generate dummy trajectories in order to confuse the adversaries. In order to confound fake trajectories and the true ones, dummy trajectories are generated under two principles: first, the movement patterns of dummies should be similar to real users; second, the intersections of trajectories should be as more as possible. Based on these rules, dummy trajectories are generated by rotating real users' trajectories.

Suppression-based method. Study in [13] is based on the assumption that different adversaries may have different and disjoint part of users' trajectories. The authors propose a suppression-based method to reduce the probability of disclosing the whole trajectories. Trajectory pieces should be suppressed when publication of these pieces may increase the whole trajectory's *breach probability* above a certain threshold.

Trajectory* k-*anonymity. In [1], Abul et al. propose a novel concept named (k,δ)-anonymity due to the imprecision of GPS data, where δ represents the possible location imprecision. Based on the concept of (k,δ)-anonymity, the authors propose an approach called *Never Walk Alone* (NWA) to achieve (k,δ)-anonymity through trajectory clustering and space translation. In [15] Yarovoy et al. address trajectory privacy problem in a quasi-identifier (QID) perspective. In this work, the authors propose a novel trajectory k-anonymity model based on an *attack graph*, then two approaches which are called *Extreme Union* and *Symmetric Union* are proposed to achieve trajectory k-anonymity. In [11] Nergiz et al. argue that since the trajectory data are published for analyzing purpose, it is useful to publish atomic trajectories rather than anonymized regions, so they design privacy protection strategies in an anonymization and reconstruction manner. First, trajectories are clustered based on *log cost metric*, each sample location on trajectories is generalized to a region containing at least k moving objects. Then trajectories are reconstructed through randomly selecting sample points from the anonymized region. More recently, [10] propose a method for achieving *true* anonymity in a trajectory dataset by transformation of the original trajectories based on spatial generalization and k-anonymity.

Our approach is different from the above mentioned ones. We are the first to formalize trajectory k-anonymity into a graph partition problem with which the information loss of the anonymization is reduced. Also, with our method, we can control the size of each partition, and systematically assign the outlier trajectories that are not partitioned into any anonymity set.

PROBLEM STATEMENTS

Trajectories of moving objects (MOBs) are collected and stored in moving object databases (MOD). For a moving object O_p, its trajectory T_p is a set of time ordered discrete location samples.

Basic Notion

In this section, we give some definitions that will be used in our study.

Definition 1. (Trajectories). A moving object O_p's trajectory is a sequence of location samples represented as T={(x_1, y_1, t_1), (x_2, y_2, t_2), ... , (x_n, y_n, t_n)}, where (x_i, y_i) represents the coordinate of O_p at time t_i for all $1 \leq i \leq n$.

Definition 2. (Synchronized Trajectories). Given two trajectories T_p and T_q, both trajectories are said to be synchronized if they have the same number of location samples at the same sampling time. A set of trajectories is said to be synchronized if all pairs of trajectories are synchronized.

If two trajectories are synchronized, location samples which have the same timestamps are called *corresponding location samples*. In fact, trajectories are not always synchronized in real-world dataset. In order to process un-synchronized trajectories, we will give an trajectory synchronization approach in section 4.

Several trajectory distance measures have been proposed in the past, we adopt the Euclidean distance measure which is

widely used in trajectory privacy-preserving community.

Definition 3. (Trajectory distance) For any two synchronized trajectories T_p and T_q, the distance between T_p and T_q is defined as the average of Euclidean distances between corresponding location samples:

$$Dist(T_p, T_q) = \frac{\sum_{i=1}^{n} \sqrt{(x_{pi} - x_{qi})^2 + (y_{pi} - y_{qi})^2}}{t_n - t_1}$$

We will use $[x_s, y_s, t_s]$ and $[x_e, y_e, t_e]$ to represent the start point and the end point of a trajectory henceforth. Locations of a trajectory's start and end can be represented as a spatial pair $[x_s, x_e]$ in x coordinate and $[y_s, y_e]$ in y coordinate. We argue that, if two trajectories have no spatio-temporal relations, it is unnecessary to compute distance between them, since they can hardly be anonymized together. For temporal relations, we mean that two trajectories should appear in a same time span; for spatial relations, it can be described by the following definition of *s-overlap*.

Definition 4. (s-overlap). Given two synchronized trajectories T_p={(x_{p1}, y_{p1}, t_1), (x_{p2}, y_{p2}, t_{p2}), ... , (x_{pn}, y_{pn}, t_n)} and T_q={(x_{q1}, y_{q1}, t_1), (x_{q2}, y_{q2}, t_2), ... , (x_{qn}, y_{qn}, t_n)} with (x_{q1}, y_{q1}, t_1) equals to (x_{qs}, y_{qs}, t_1) and (x_{qn}, y_{qn}, t_n) equals to (x_{qe}, y_{qe}, t_n), if $\exists$ $(x_i, y_i) \in T_p$ (i=1, 2, ..., n), $x_i \in [x_{qs}, x_{qe}]$ or $y_i \in [y_{qs}, y_{qe}]$, T_p *s-overlaps* with T_q, where s>0 is the portion of T_p that s-overlaps with T_q.

S-overlap describes the spatial relationships between trajectories, it will be employed in trajectory graph construction in section 4 to reduce edges of a graph.

Attack Model Assumption

There are two important assumptions to declare before defining the attack models: firstly, the public spatio-temporal space is accessible to any observers no matter he is an adversary or not; secondly, the users which the attackers are trying to re-identify is unknown to the adversary. Before our methods, we assume the traces are already anonymized by replacing the true identifier with a random and unique pseudonym, we call this phase tentative anonymous. We found that after tentative anonymous, threatens of privacy leakage still exist through following attacks:

- ***Linkage attack***: Users may expose its whereabouts through background information, such as a causal talk between two people about one's work place may expose the destination of his everyday trajectories. Suppose an adversary knows a user O_p will be at location L_i at time t_i through background information linkage, while t_i happens to be a sample time points in O_p's trajectory in the published dataset D^*. The adversary tries to link this information D^* to re-identify the whole trajectory of O_p. This kind of attack is called ***linkage attack***. In this scenario, we assume background information happens to be at sample time and all the background information is trusted.

- ***Observation attack***: Suppose an adversary who stays in one place or moving through a pre-defined movement strategy to observe users, we assume that there is no noise during the observation. When the adversary found O_p appearing, he observes O_p for at least one sample period in order to identify O_p's precise location at sample time. In the worst case, if O_p is the only person appearing at that place at that time, the adversary can re-identify O_p's whole trajectory history from the tentative anonymized dataset D^* with 100% confidence. If there are N users appearing in one place at the same time, the re-identification possibility drops to $1/N$ if the adversary has no other observations or background information. This kind of attack is called ***observation attack***. It should be noted that adversaries in observation attack do not purely mean *persons*, it also contains various data collectors, such as cameras in a shopping mall, or monitors on roadside.

Problem Characterization

In order to withstand both attacks, we must adopt a more strict privacy protection strategy rather than only removing the true identifiers. Trajectory k-anonymity works with both of the two attack models. For linkage attack, the linkage of a location to a specific user will not help adversaries to re-identify a user since the published trajectories are discrete regions shared by k users, the same can be seen in observation attack, even if the adversaries observe O_p himself somewhere, O_p still can not be re-identified since the published trajectories are generalized to regions of location samples. However, anonymizing k trajectories together may cause serious information loss, so the key issue in trajectory k-anonymity is how to find k trajectories that can be anonymized together with minimum information loss. In our proposed method, we try to formalize trajectory k-anonymity to a graph partition problem, then design a greedy method to form trajectory k-anonymity set.

Definition 5. (Trajectory k-anonymity). Given a moving object database D, a published database D^* is an anonymized version of D. The resulting anonymization must meet the following requirements:

- Each trajectory in D^* is anonymized with other k-1 trajectories on all time samples;

- The published version D^* should have minimum information loss.

For both two attack models, the breach probability of trajectory k-anonymity is under $1/k$ even if adversaries know how many trajectories are anonymized together and their distributions in each anonymized region.

PROPOSED SOLUTIONS

Our proposed method consists of three main phases: phase I, pre-processing of trajectories to form equivalent classes with the same time span; phase II, constructing trajectory graphs for each equivalent class based on the notion of trajectory *s-overlap*; phase III, partitioning trajectory graphs, connected components of size k(or larger than k) are retained to form trajectory anonymity sets. At last, each location sample on trajectories are generalized to a region containing at least k

Notations	Explanations
O_p	A moving object in our dataset.
T_p	Trajectory of moving object O_p.
k	Users' anonymity requirement.
$TG(V, E)$	Trajectory graph with V represents vertices and E represents edges.
EC	The equivalent class.
$\omega_{i,j}$	the weight of edge (v_i, v_j).
V_i	A partition of TG with vertex number larger than k.
D_i	A partition of TG with vertex number less than k.
$T_p.x_s$	x coordinate of T_p's start point.
$T_p.x_e$	x coordinate of T_p's end point.
th_t	Threshold to recognize a stay.

Table 1. A list of notations

users. Table 1 shows a partial list of symbols used in the following.

Pre-processing

In trajectory data pre-processing phase, three tasks should be complemented: start/end points recognition; equivalent class formation and trajectory synchronization.

A trajectory of a moving object may contain several long stays during his way. For example, a person arrives at home on someday's afternoon and leaves home on next morning. This long stay split the whole trajectory into two sperate parts. So the start and end points of a trajectory should be recognized. For a trajectory of a moving object O_p, if a stay in the trajectory lasts for a time longer than a given threshold th_t, it is regarded as the end point of the trajectory, while the next triple is assigned as a start point of a new trajectory of O_p.

Another concern in pre-processing phase is that, trajectories may begin or end at arbitrary time. However, trajectory k-anonymity only works when trajectories are in the same time span. In order to enforce large equivalent classes within the same time span, we adopt a new strategy to form equivalent classes. Trajectories with start and end time within a time interval $[td_{s1}, td_{s2}]$ and $[td_{e1}, td_{e2}]$ respectively are assigned into a same equivalent class. A more formal definition is given in the following.

Definition 6. (Trajectory Equivalent Class). Trajectory equivalent classes consist of trajectories within nearly the same time span. Given two trajectories T_p and T_q, both of them are assigned in a same equivalent class if and only if $T_p.t_s, T_q.t_s \in [td_{s1}, td_{s2}]$ and $T_p.t_e, T_q.t_e \in [td_{e1}, td_{e2}]$.

Start time and end time of trajectories in a same equivalent class must happen in time intervals $[td_{s1}, td_{s2}]$ and $[td_{e1}, td_{e2}]$ respectively according to the definition. For example, time interval $[td_{s1}, td_{s2}]$ can be set as [08:00:00, 08:10:00] and $[td_{e1}, td_{e2}]$ can be set as [09:00:00, 09:10:00], that is to say, trajectories that start during [08:00:00, 08:10:00] and end during [09:00:00, 09:10:00] of a same day can be assigned into an equivalent class. Time intervals $[td_{s1}, td_{s2}]$ and $[td_{e1}, td_{e2}]$ are set according to the characteristics of the dataset. If the data is sparse, time intervals should be set broader and vice versa.

Trajectories in the same equivalent class can then be synchronized through inserting new location samples. We assume that between two sample locations of a trajectory, the object is moving along a straight line in a constant speed. Given two trajectories T_p and T_q, if a sample location in T_p has a timestamp t_s which is not in T_q, then insert a new sample location $[x_s, y_s]$ to T_q having the timestamp t_s and $[x_s, y_s]$ is between two existing consequent locations.

After pre-processing, trajectories in equivalent classes are synchronized trajectories within the same time span.

Algorithm 1: **TrjGraphCons**(EC)

Input: Trajectory equivalent class EC=$T_1, T_2, \ldots, T_n$
Output: $TG = (V, E)$// Vertexes and edges of trajectory graph TG;

```
1:  E ← Φ; V ← T_1;
2:  V_left ← EC − V;
3:    while (V_left is not empty) do
4:      if T_i in V_left s-overlap with trajectories in V
5:        for each trajectory T_j s-overlap with T_i do
6:              ω_ij ← distance(T_i, T_j);
7:              E ← (T_i, T_j, ω_ij);
8:              V ← V + T_j;
9:              V_left ← V_left − V_i;
10:         end for
11:       else
12:         V ← V + Ti;
13:         V_left ← V − T_i;
14:   end while
15: end
```

Trajectory Graph Construction

In this section, we first define *trajectory graph*, then we show procedure of trajectory graph construction.

Definition 7. (Trajectory Graph). A trajectory graph $TG = (V, E)$ is a weighted undirected graph where vertices represent trajectories; an edge exists between two vertices T_p and T_q if they s-overlap with each other with s>0; the weight of the edge (T_p, T_q) is the trajectory distance between T_p and T_q.

Algorithm 1 shows the procedure of trajectory graph construction. The input of the algorithm is trajectory equivalent class EC, while the output is a trajectory graph TG=(V, E). At the beginning, the edges set E is empty, and the vertices set V contains an arbitrary trajectory in EC, trajectories that are not included by TG are put into V_{left} (line 1-2). Then each trajectory in V_{left} is scanned to check if any trajectory s-overlap with trajectories in V. If *s-overlap* happens, the distance of the overlapped trajectories is computed and stored as weight of the edge, the vertices is then removed from V_{left} and put into V(line 5-10); if *s-overlap* doesn't happen, just

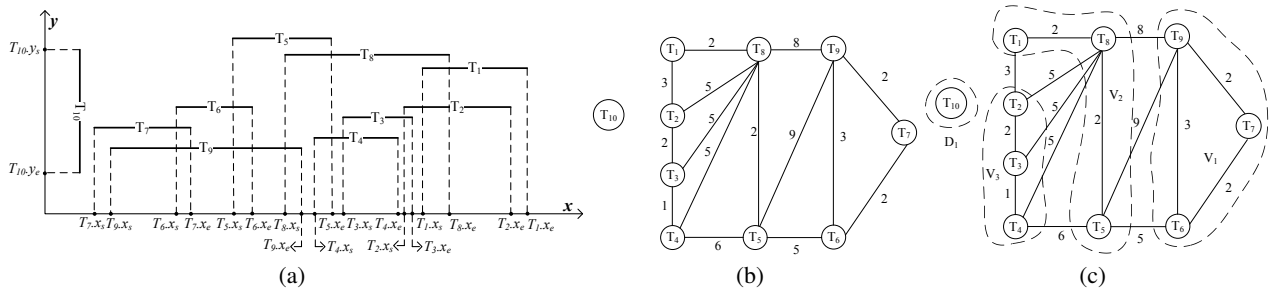


Figure 1. (a) shows trajectories distribution, (b) is a trajectory graph based on (a), (c) is a partition of the graph with 3 trajectories contained in each partition.

move the trajectory from V_{left} to V(line 11-13).

Trajectory graph is very likely to be an un-connected graph with several connected components since some trajectories may not s-overlap with others. If two trajectories do not s-overlap, there is no edge between them, this can reduce edges in a trajectory graph.

Trajectory Anonymization

We then turn trajectory k-anonymity problem to a graph partition one in order to control the size of each anonymity set and assign the outliers systematically. The most ideal situation is that each partition is a connected sub-graph of k vertices with minimized cost, which satisfies the requirement of trajectory k-anonymity. In this section, we show a running example of graph partition-based trajectory k-anonymity, then we formally define the problem as a k-node partition. Afterwards, an approximate approach is given to solve the problem.

A running example

A running example of the trajectory k-anonymity problem is shown in figure 1. Figure 1.(a) shows the distribution of ten trajectories in an equivalent class, where $T_i.x_s$ represents the start point's x-coordinate of trajectory T_i, while the $T_i.x_e$ represents the end point's x-coordinate, the same can be seen in y-coordinate. A trajectory may have projections on both x-coordinate and y-coordinate, for simplification, the example shown in figure 1 considered projections in one direction only. A trajectory graph in figure1.(b) is constructed according to trajectories in figure1.(a), where vertices are trajectories, while weights of edges represents the distances between two adjacent vertices. T_{10} is not connected to the component formed by T_1-T_9, since it does not *s-overlap* with any trajectory in that component. For a k-anonymity requirement where k=3, TG is partitioned into four disjoint component V_1, V_2, V_3 and D_1, as for partitions V_1, V_2 and V_3, each of them contains three nodes, and the sum of weights in each component is minimized. D_1 is dropped since it does not satisfy the 3-anonymity requirement. V_1, V_2 and V_3 are trajectory k-anonymity sets with k=3.

k-node partition

Different from the k-way partition [8], our problem is defined as a graph k-node partition problem. The definition is shown in the following.

Definition 8. (k-node partition) Given a trajectory graph TG, a k-node partition of TG is a disjoint connected component V_1, V_2, ..., V_l and D_1, D_2, ..., D_h where:

- $V_1 \cup V_2 \cup \ldots V_l \cup D_1 \cup D_2 \cup \ldots \cup D_h = TG$;
- for any i, $k \leq |V_i| \leq 2k$-1 and $|D_i| < k$;
- for any i and j, $V_i \cap V_j = \phi$; $D_i \cap D_j = \phi$ and $D_i \cap V_j = \phi$;
- W_i represents sum of edge weights belongs to a same partition, $CostTG = \sum_{i=1}^{l} W_i$ is minimized, while h is as small as possible.

In k-node partition, V_1, V_2, ..., V_l form the k anonymity sets with each subset contains at least k vertices, while D_1, D_2, ..., D_h represent the partitions that will be dropped, since each of them contains less than k vertices which fails to satisfy the minimum privacy requirements. The fourth condition requires that the number of dropped partitions should be as small as possible. We also analyzed the generation of drop sets D_i, which may include the following three reasons: firstly, TG is an un-connected graph which may contain several connected components, if a connected component has less than k nodes, neither of its vertices can be partitioned into any sub graph, so it should be a drop set; secondly, in a connected component, the deletion of several k-node partition may split TG into smaller components, maybe with vertex number less than k; thirdly, if the vertex number n satisfies (n mod k) $\neq$ 0, it will also cause extra trajectories left.

In order to minimize information loss, the k-node partition requires that, the *internal cost* which means the sum of edge weights belongs to a same partition is minimized and the number of drop sets should be as small as possible. We then prove k-node partition problem as *NP-complete*.

LEMMA 1. *The k-node partition problem is NP-complete.*

PROOF. We prove the lemma by reducing our problem to k-way partition problem which is well known *NP-complete* [4]. An instance of a k-way partition problem is that given a graph $G = (V, E)$ with $|V|=n$, a partition of G is a set of nonempty non-overlap component of G, V_1, V_2, ... ,V_k, such that: $V_1 \cup V_2 \cup \ldots V_k = G$ and the *external cost* $\sum_{i=1}^{n} \omega_i$ is minimized, where ω_i represents the weight of edges whose incident vertices belong to different subsets.

Algorithm 2: Greedy k-node partition(TG, k)

Input: A trajectory graph TG, node number in a partition k;
Output: Two sets of partitions RS and DS;
1: **for each** ($G_i \in TG$ and $|G_i| > k$) **do**// G_i is a connected component in TG
2: $C \leftarrow$ GreedyFindC(G_i, k);
3: $TG = TG - C$;
4: $RS \leftarrow C$;
5: $DS \leftarrow$ components with node number less than k;
6: **end for**
7: **if** $|G_i| < k$
8: $DS \leftarrow G_i$;
9: **endif**

To reduce this problem to the k-node partition problem, we observe that given a graph $G = (V, E)$, the sum of weights of all edges is fixed, the minimization of external cost means the maximization of internal cost. Apparently, a constant N and a series of number μ_i can be found, where $\omega_i = (N - \mu_i)$ for all edges in TG, thus the definition in k-way partition: maximization of internal cost $\sum_{i=1}^{n}(N - \mu_i)$ turns to be minimization of $\sum_{i=1}^{n}\mu_i$. We then turn μ_i to ω_i in our settings, which is exactly the description of our k-node partition problem. Moreover, k-node partition requires that each resulted partition is connected while k-way partition does not. So the k-node partition is even harder than the k-way partition. This completes the proof. □

Since the k-node partition problem is NP-complete, our goal in is paper is try to find an approximate method to partition the trajectory graph.

Algorithms

TG is a trajectory graph of n vertices, a k-node partition will be achieved by Algorithm 2. For each connected component in TG, the greedy method *GreedyFindC* is called until TG has no connected components that is larger than k. In *GreedyFindC* (Algorithm 3), we adopt a greedy strategy to partition TG. In algorithm 2, two sets of connected components are maintained, one is retain set (RS) and the other is drop set (DS). Partitions with k vertices are kept in RS, while partitions with less than k vertices are kept in DS.

How to greedily find a partition C in a connected component of TG is represented in Algorithm 3. The main idea of the algorithm is to greedily find the edges with minimum edge-weight to form a partition unless the deletion causes generation of components with vertex number less than k. The algorithm starts from an edge with minimum edge-weight, two vertices affiliated with the edge formed a component C, the vertices are pushed into a stack SC (line 2-3). All the edges adjacent to C is put into X (line 4). While $|C| < k$, C spread itself by finding the vertices which connect it with minimum edge-weight, then including that vertex into C (line 5-11). While $|C| = k$, we try to delete it from G_i. Before the deletion, we need to check if the deletion causes generation of connected components with vertex number less than k. If so, redo the detection-deletion phase to interchange another vertex to see if connected component in $TG - C$ have more than

Algorithm 3: GreedyFindC (G_i, k)

Input: A connected component in G_i=(V,E), vertex number of each partition k;
Output: a partition C with k nodes;
1: Token=0; X=Φ;
2: Find minimum weight edge $e = (v_s, v_e)$, put v_s, v_e in C;
3: Push(v_s, SC), Push(SC, v_e);// *SC is a stack*
4: $X \leftarrow$ edge x which is adjacent to node in C;
5: **While** $|C| < k$ **do**
6: **if** (X = Φ) then
7: $X \leftarrow$ edge x which is adjacent to node in C;
8: **end if**
9: Find minimum weight edge e=(v_i, v_j) in X;
10: delete e from X;
11: $C \leftarrow v_j$; Push(SC, v_j);
12: $X \leftarrow$ edges adjacent to C and not covered by C;
13: **if** $|C| = k$ and token=0 then
14: $G_i \leftarrow$ delete-detection(G_i,C)
15: **if** exist $GS_i \in G_i$ and $|GS_i| < k$ then
16: $C \leftarrow C$-pop(SC);
17: delete connected edges in X;
18: **if** X contains only (v_s, v_e) then
19: token=1;
20: Delete nodes in C until it contains (v_s, v_e)
21: **end if**
22: **else**
23: $G_i \leftarrow G_i - C$;
24: return C;
25: **end if**
26: **else if** $|C| \geq k$ and token=1 then
27: $G_i \leftarrow G_i - C$;
28: return C ;
29: **end if**
30: **end while**

k vertices (line 13-17). If all the interchanges cannot work, C goes back to its original choice by deleting the vertices of C (line 18-21), and do the *while* loop again. At last, C is returned and deleted from G_i.

Each partition in RS is a k-anonymity set of trajectories and partitions in DS are dropped. At last, $D*$ is published with each trajectory anonymized with other k-1 trajectories.

Measure of Privacy and Information Loss

The Greedy k-node partition algorithm achieves trajectory k-anonymity based on graph partition, so the breach probability is under $1/k$ even if the adversary knows the number of k and locations of each moving object that anonymized together. If the adversaries have no knowledge about the privacy model, the breach probability drops to $1/area$, where $area$ represents the area size of the generalized region.

Here we measure information loss by the sum of area size of anonymized region, while the number of dropped trajectories is also considered in this measure. The computation of information loss is shown in the following formula.

Dataset	$\|D\|$	$\|D_{ec}\|$	$MaxNo$	$MinNo$	$Ratio$
TRUCKS	5587	154	340	1	32.96%
OLDEN.	26471	132	388	72	13.31%

Table 2. Datasets used in experiments

$$InfoLoss = \sum_{i=1}^{l} \sum_{j=1}^{m} \frac{Area(x_j, y_j, t_j)}{MaxArea} + \sum_{i=1}^{h} |T_i|$$

Information loss are mainly caused for two reasons: first, generalization of a location to a region makes the data utility shrinks; second, the deletion of trajectories makes the utility of a data item totally lost. In the formula, l represents the number of partitions with k or more than k vertices, while h represents the number of partitions with less than k vertices. $Area(x_j, y_j, t_j)$ means the area size of generalized regions of location (x_j, y_j) at time t_j, $MaxArea$ means the area size of the whole space. If a trajectory is deleted, each location on the trajectory is lost, meaning that $Area(x_j, y_j, t_j)$ equals to $MaxArea$, so the total shrink of utility of trajectory T_i turns to be $|T_i|$, the number of location samples of trajectory T_i.

EXPERIMENTS

In this section we report the empirical evaluation of our proposed method. The evaluations on both real-word dataset and the synthetic one are reported.

Experimental Data

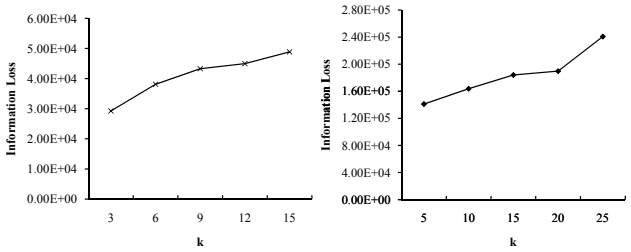
Both real-world and synthetic trajectory dataset are used in our evaluation. The real-world dataset, called TRUCKS [5] henceforth, contains trajectories of 50 trucks delivering concrete to several construction places around Athens metropolitan area in Greece for 33 distinct days. The second dataset, called OLDENBURG henceforth has been generated using Brinkhoff's generator [2]. OLDENBURG contains 10,000 moving objects which represents one day movement over the city Oldenburg in Germany. The generator parameters are 150 timestamps and speed 50.

In table 2, we report the characteristics and some results after pre-processing of both datasets. $|D|$ represents the number of trajectories in each dataset after pre-processing; $|D_{ec}|$ is the number of equivalent classes in each dataset; $MaxNo$ and $MinNo$ present the maximum and minimum population of equivalent classes respectively; while $Ratio$ represents the average ratio of s-overlap between trajectories in each equivalent class. It can be seen that, TRUCKS is much sparser than OLDENBURG, that means TRUCKS contains smaller number of trajectories distributed over a large spatial space.

Data Utility

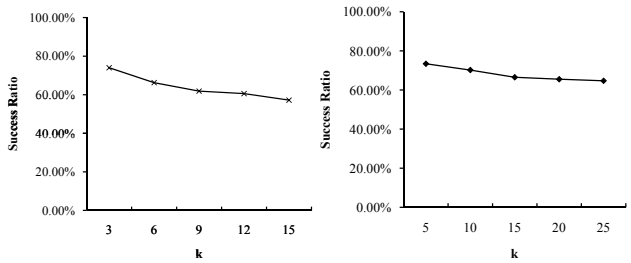
We next compare greedy k-node partition algorithm on both real-world and synthetic datasets in terms of data utility. In the following we discuss the measure that we adopt.

Information loss: The information loss measure is defined in section 4. We measure information loss on both OLDENBURG and TRUCKS. The results are shown in figure 2. Since TRUCKS is much sparser than OLDENBURG, we set different k values for two datasets. Information loss grows as the privacy parameter k grows, that is because larger k value causes larger area size of the generalized region. In another aspect, the reduction of success ratio as k grows also cases more information loss. It can be seen that, the growth of information loss is relatively stable, meaning that increasing of privacy level won't cause sharply increase of information loss.



(a) TRUCKS (b) OLDENBURG

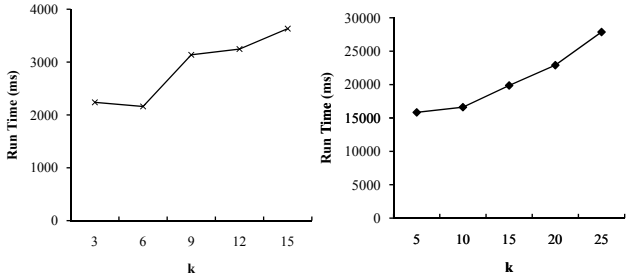
Figure 2. Information loss evaluation



(a) TRUCKS (b) OLDENBURG

Figure 3. Success ratio evaluation

Success Ratio: The success ratio measures the portion of successfully anonymized trajectories. We measure success ratio on both OLDENBURG and TRUCKS, the results are shown in figure 3. It can be seen that, success ratio of OLDENBURG is above 65%, although success ratio drops as k grows. For TRUCKS, success ratio is above 60%, less than that on OLDENBURG, that is because TRUCKS is sparser than OLDENBURG, making the partition of k vertices sometimes hard to achieve.



(a) TRUCKS (b) OLDENBURG

Figure 4. Run time evaluation

Efficiency

The greedy k-node partition algorithm is implemented in JAVA, and all experiments are performed on a Intel Core 2 Quad 2.66G processor with 4GB RAM over a windows 7 platform.

The algorithm requires 13,897 ms in average to pre-process OLDENBURG, and 194,732 ms to construct 132 trajectory graphs for all the equivalent classes on OLDENBURG. For TRUCKS, pre-processing cost 10,072 ms and a total 5455 ms to build 154 trajectory graphs. Running time of greedy k-node partition are shown in figure 4. Since TRUCKS is much smaller than OLDENBURG in size, greedy k-node partition runs much faster on TRUCKS than on OLDENBURG. Running time on both datasets grows as parameter k grows, that is because growth of k may cause more cost on deletion-detection phase.

CONCLUSIONS

Collection and publication of location based service users' everyday trajectories poses serious problems to users' personal privacy. In this paper, we analyze two kinds of attacks on tentative anonymized dataset, then we formalize trajectory k-anonymity problem into a k-node graph partition problem. We prove the k-node partition is NP-complete. Afterwards a greedy method to achieve k-node partition is proposed in this paper to get an approximate result. At last, a series of experimental evaluations are presented. Our proposed method reduces information loss while providing high privacy guarantees. In the future, we will concentrate on how to optimize the k-node partition methods as well as adopting other trajectory distance measures.

ACKNOWLEDGMENTS

This research was partially supported by the grants from the Natural Science Foundation of China (No.60833005, 61070055, 91024032), the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China (No.10XNI018), National Science and Technology Major Project of Key Electronic Devices, High-end General-purpose Chips and Fundamental Software Products (No. 2010ZX01042-002-003), Specialized Research Fund for the Doctoral Program of Higher Education of China(No. 200800020002).

REFERENCES

1. O. Abul, F. Bonchi, and M. Nanni. Never walk alone: uncertainty for anonymity in moving objects databases. In *Proc. ICDE 2008*, pages 376–385, 2008.

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

3. X. Cao, G. Cong, and C. S. Jensen. Mining significant semantic locations from gps data. In *Proc. VLDB 2010*, pages 1009–1020, 2010.

4. C. M. Fiduccia and R. M. Mattheyses. A liner time heuristic for improving network partition. In *Proc. DAC 1982*, pages 175–181, 1982.

5. E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis. Nearest neighbor search on moving object trajectories. In *Proc. SSTD 2005*, pages 328–345, 2005.

6. B. Gedik and L. Liu. Location privacy in mobile systems: A personalized anonymization model. In *Proc. ICDSC 2005*, pages 376–385, 2005.

7. M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proc. of MobiSys 2003*, pages 31–42, 2003.

8. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 1970.

9. H. Kido, Y. Yanagisawa, and T. Satoh. Protection of location privacy using dummies for location-based services. In *Proc.ICDEW 2005*, 2005.

10. A. Moreale, G. Andrienko, N. Andrienko, F. Giannotti, D. Pedreschi, S, Rinzivillo, and S. Wrobel. Movement data anonymity through generalization. *IEEE Transactions on Data Privacy*, 3:91–121, 2010.

11. M. E. Nergizand, M. Atzori, Y. Saygin, and G. Baris. Towards trajectory anonymization: a generalization-based approach. *IEEE Transactions on Data Privacy*, 2:47–75, 2009.

12. X. Pan, X. Meng, and J. Xu. Distortion-based anonymity for continuous queries in location-based mobile services. In *Proc. GIS 2009*, pages 256–265, 2009.

13. M. Terrovitis and N. Mamoulis. Privacy preserving in the publication of trajectories. In *Proc. MDM 2008*, pages 65–72, 2008.

14. Z. Yan. Towards semantic trajectory data analysis: a conceptual and computational approach. In *Proc. VLDB 2009 PhD Workshop*, 2009.

15. R. Yarovoy, F. Bonchi, S. Lakshmananand, and W. H. Wang. Anonymizing moving objects: How to hide a mob in a crowd? In *Proc. EDBT 2009*, pages 72–83, 2009.

16. T. H. You, W. C. Peng, and W. C. Lee. Protecting moving trajectories with dummies. In *Proc. MDM 2007*, pages 278–282, 2007.

17. Y. Zheng, L. Zhang, X. Xie, and W. Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proc. WWW 2009*, pages 791–800, 2009.