Efficient Privacy-Preserving Approaches for Trajectory Datasets

Md Yeakub Hassan, Ullash Saha, Noman Mohammed, Stephane Durocher, Avery Miller
Department of Computer Science, University of Manitoba, Winnipeg, Manitoba
Email: {hassanmy, sahau, noman, durocher, amiller}@cs.umanitoba.ca

***Abstract*—The use of credit cards or RFID cards for transaction payment has become ubiquitous in recent years. This transactional data is stored in the form of a trajectory, including a sequence of locations at which the customer used the credit card. Such datasets are sensitive since they store customer's locations and purchasing patterns. Releasing this type of dataset without proper anonymization may breach individual's privacy. Moreover, an adversary can attack the dataset with partial trajectory knowledge to reveal sensitive information about a person. We study trajectory data anonymization techniques for this problem, where a data publisher can construct a safe dataset with minimum information loss. A global suppression algorithm was proposed in [1] for trajectory data anonymization. The algorithm is computationally expensive for large trajectory datasets. We propose a tree-based data structure that significantly reduces the computational cost of the global suppression algorithm. In an experiment with a real-world dataset, our data structure performs global suppression in 50% less time than the state-of-the-art algorithm.**

***Keywords*-anonymizaton; trajectory; generalization; suppression; projection; tree**

I. Introduction

In recent years, the technology of tracking moving objects using GPS and other location-based services has developed rapidly. This type of movement data is stored in the form of trajectories and often refers to a sequence of time-stamped locations [2]. As technology becomes cheaper and more precise, a large amount of trajectory data is generated from mobile handsets, navigation devices, and other IoT devices. Trajectory data mining is important for many real-world applications such as homeland security, law enforcement, weather forecasting, recommendation systems, personalized advertisements, and so [3]. Moreover, many service providers collect this trajectory data to provide location-based services and advertising. However, this trajectory data may contain personal and sensitive information of the user. Therefore, sharing this data among different parties is a concern of privacy leakage of the users. An attacker could easily identify an individual from the trajectory datasets if the proper privacy policy is not imposed.

Suppose that a company $Tap\&Pay$ (like a credit card company or a bank) provides electronic money cards (credit/debit) to its customers. These types of cards allow cashless payments at most retailers. When a customer makes a transaction using this credit/debit card in a physical location, that location is stored into the database of the credit card company, along with other sensitive information about the user and the transaction. Therefore, companies like $Tap\&Pay$ daily accumulate a large of amount of transactional data of their customers. However, this transactional data reveals customer's trajectories with their movements and behavioral patterns over a sequence of different geographic locations. $Tap\&Pay$ and similar companies are often bound to publish or share this trajectory dataset with the companies or stores that collaborate with them [1], [4]. Therefore, the major concern of a data publisher is to protect customer privacy while publishing this trajectory dataset. Publishing this dataset even after hiding customer identities may disclose individuals privacy. A store having the $Tap\&Pay$ payment option has access to its own records of the transactions that occur in that store. A company may have multiple stores, and the transactional data of a customer from these stores can be connected to obtain partial trajectory data of the customer. This partial trajectory of a customer can be mapped into a published dataset, allowing an attacker to reveal a customer's masked identity, and to identify the complete trajectory of purchases and stores visited by that customer. In this problem, the attackers or adversaries are the companies that collaborate with $Tap\&Pay$, and the data owner (i.e., $Tap\&Pay$) knows what partial trajectory database each attacker has.

Table I: Published Dataset T (left) and Company B's Dataset T^B (right)

id	trajectory
t_1	$a_1 \rightarrow a_2 \rightarrow b_2$
t_2	$b_2 \rightarrow a_2 \rightarrow a_1 \rightarrow c_1$
t_3	$b_2 \rightarrow c_1 \rightarrow b_1 \rightarrow a_2 \rightarrow b_2$
t_4	$c_1 \rightarrow a_1 \rightarrow a_1 \rightarrow b_2$

id	trajectory
t_1^B	b_2
t_2^B	b_2
t_3^B	$b_2 \rightarrow b_1 \rightarrow b_2$
t_4^B	b_2

Consider the dataset T in Table I which is published by the data owner company $Tap\&Pay$. Trajectories are represented as a sequence of store locations of different companies. We assume three companies A, B, C use $Tap\&Pay$ payment options in their stores. The stores owned by companies A, B and C are $\{a_1, a_2\}, \{b_1, b_2\}$, and $\{c_1\}$, respectively. Table I represents the dataset T^B owned by

company B. Moreover, company B can get T^B from the transaction histories of their different stores. Here, company B can play the role of an attacker to map the dataset T^B with the published dataset T (ignoring the identity column) to obtain information about additional stores visited by its customers. As an example, trajectory t_3^B can be matched with the original trajectory t_3 and company B learns that this customer has been visiting the stores c_1 and a_2. Nevertheless, data publishers in this case know what part of dataset an attacker has and they can use it to anonymize the original dataset before its publication. In trajectory data anonymization, suppression is a process of removing selected portions from a trajectory in order to protect its privacy. Global and local suppression algorithms were proposed in the literature for this particular privacy problem [1]. In global suppression, a selected sub-trajectory is globally suppressed from all the trajectories. The local suppression suppresses a sub-trajectory from a single trajectory at a time. In [1], the algorithms first check if a dataset leaks the privacy and how much anonymization is needed to protect privacy. Finally, the privacy issues are resolved with minimum information loss. However, these algorithms are computationally expensive and take significant time to execute on trajectory datasets.

In this paper, we propose a novel tree-based data structure that significantly reduces the computational cost of the existing global suppression algorithm. Our approach also produces the correct anonymization for all cases. Simulation results show that our proposed approach provides better performance than the state-of-the-art global suppression algorithm in terms of running time. The major contributions of this paper are summarized below:

- We propose a novel data structure to store the trajectories. The global suppression algorithm [1] is implemented using the proposed data structure.
- The proposed data structure reduces the running time of the global suppression algorithm [1]. Moreover, an efficient update operation in the data structure ensures correct anonymization results.
- We run an experiment with a real-world dataset in which the proposed data structure requires 50% less time to compute the anonymized dataset than the global suppression algorithm [1].

The rest of the paper is organized as follows. In Section II, previous related work is discussed. The problem formulation with necessary notation and definitions is presented in Section III. The most relevant existing work on this problem is briefly discussed in Section IV. Our proposed approach is discussed in Section V. Experimental results of our proposed approach are discussed in Section VI. Finally, we conclude with a discussion of possible directions for future work in Section VII.

II. Related Work

Numerous research has been conducted on privacy preservation techniques for set-valued datasets. Credit card transactions are stored in the form of set-valued data which represents a customer's trajectory. Privacy preservation techniques on this type of trajectory dataset have been studied over the past few years. Various techniques, such as location generalization, cryptography or differential privacy, were explored for location privacy protection [5].

Clustering and partition-based data sanitization techniques were well studied for location-based data anonymization [6], [7]. Abul et al. [7] proposed a (k, δ)-anonymity technique to protect the location privacy of a trajectory dataset of moving objects, where the dataset is divided into cylindrical volumes of radius δ. Trujillo-Rasua and Domingo-Ferrer [6] showed the limitation of (k, δ)-anonymity when $\delta > 0$. Moreover, (k, δ)-anonymity does not fulfill trajectory k-anonymity when δ is greater than 0. Differential privacy is another method of data anonymization that adds noise to the synthetic dataset by keeping most of the attributes of the original datasets to ensure privacy protection [8]. This differential privacy technique was also implemented on trajectory datasets [9]–[11]. Jiang et al. [9] used differential privacy for coordinate-based trajectory datasets. Bonomi et al. [11] used prefix-trees for mining sequential patterns that can be used to identify the location patterns of the data summary. He et al. [10] used a hierarchical reference system to synthesize the raw trajectory data that guarantees strong privacy in terms of ϵ-differential privacy. Hikita and Yamaguchi [12] proposed a population-based anonymization approach for trajectory datasets. They used the quad-tree based approach to divide the geographical area and implemented dynamic method to adjust the cluster size. Though differential privacy ensures the greater utility of the data while improving the privacy [8], we are not concerned about the privacy policy in this paper.

Generalization and suppression approaches were also imposed to protect the privacy of trajectory datasets [13]–[16]. Nergiz et al. [13] first proposed a generalization-based approach for trajectory data anonymization. They formulated the concept of classical k-anonymity in their generalization-based approach. Terrovitis et al. [14], [15] introduced a new version of k-anonymity, called k^m anonymity which is used to anonymize the set-valued data. This k^m anonymity reduces the curse of dimensionality of set valued datasets. Chen et al. [16] mentioned some difficulties which make the anonymization harder in higher dimensional and sparse datasets. They also introduced a local suppression called the $(K, C)_L$ model to anonymize the trajectory data for preventing both identity and attribute linkage attack. Recently, Terrovitis et al. [1] deployed global suppression and splitting methods to protect the privacy of trajectory datasets where the trajectories are the sequence of credit card transaction

locations of a customer. They adopted the concept of l-diversity in their approaches. However, these algorithms follow a greedy strategy which falls into the problem of locality. An iterative search was applied on the whole dataset where they identify whether there are any privacy threats or not. But in real-world applications, trajectory datasets contains a large number of higher-dimensional data, which makes their algorithms computationally expensive. Lin et al. [4] implemented tree-based architecture to store large trajectory datasets in an efficient way to reduce the running time of the global suppression algorithm proposed in [1]. However, their proposed data structure has some faults which produce an incorrect solution in some cases. The update operation in the tree does not give a correct tree for the global suppression. Hence, the iterative procedure of global suppression produces an incorrect solution. In this paper, we propose a novel tree-based data structure that stores the whole dataset in an efficient way that takes less time on average to access and update the tree. Moreover, our proposed data structure and technique give the correct solution in all cases.

III. Preliminaries

In this section we provide the necessary definitions and notations that have been used throughout the paper. The following definitions and notations are similar to the privacy model proposed in [1].

*Definition 3.1: Given a set of locations $\mathcal{L}$, a **trajectory** $t = \{l_1, l_2, \ldots, l_n\}$ is a sequence of n locations, where each location $l_i \in \mathcal{L}$.*

These locations denote sequence of retail locations where a given customer uses $Tap\&Pay$ card to make a transaction. Let $\mathcal{A}$ be a set of adversaries, i.e, data owner (e.g., a store chain). The set of locations owned by an adversary $A \in \mathcal{A}$ is denoted as $\mathcal{L}^A \in \mathcal{L}$. For a published dataset T, the knowledge of an adversary $A \in \mathcal{A}$ is denoted T^A.

*Definition 3.2: The **projection** of a trajectory $t \in T$ to an adversary A, denoted by $\pi_A(t)$, is a sub-trajectory of t that contains only the locations that belong to $\mathcal{L}^A$.*

In Table I, the projection of t_1 to adversary A is a $\pi_A(t_1) = \{a_1, a_2\}$. Hence, the knowledge of an adversary A for published dataset T is the projection of every trajectory $t \in T$ to A. Therefore, the adversary's dataset is defined as $T^A = \{\pi_A(t) : t \in T\}$.

*Definition 3.3: For any trajectory $t^A \in T^A$ in adversary's dataset, a trajectory $t \in T$ is in the **support set** of t^A if $t^A = \pi_A(t)$. The support set of t^A for the published dataset T is defined as $S_T(t^A) = \{t \in T : \pi_A(t) = t^A\}$.*

In Table I, the support of t_1^B is $S_T(t_1^B) = \{t_1, t_2, t_4\}$. Thus, for any projection $\pi_A(t)$, the support set is a set of trajectories that contain $\pi_A(t)$. Here the number of trajectories in $S_T(t^A)$ containing a location $\lambda \notin \mathcal{L}^A$ is denoted as $n@S_T(\lambda, t^A)$. Using a customer's partial trajectory from a company's (adversary's) dataset, now we calculate the probability of the customer visiting a location which is not owned by that company. For a customer's trajectory $t^A \in T^A$, the probability that they visit in a location $\lambda \notin \mathcal{L}^A$ is defined as

$$P_T(\lambda, t^A) = \frac{n@S_T(\lambda, t^A)}{|S_T(t^A)|}. \tag{1}$$

*Definition 3.4: For a given probability threshold P_{br}, a pair (λ, t^A) is called a **problematic pair** if $P_T(\lambda, t^A) > P_{br}$. A trajectory $t^A \in T^A$ is problematic if it creates at least one problematic pair with any $\lambda \notin \mathcal{L}^A$. The number of problems for problematic pair (λ, t^A) is equal to the number of trajectories in the support $S_T(t^A)$ that contain λ*

In Table I, $P_T(a_2, t_1^B)$ is $\frac{2}{3}$. If P_{br} is 0.5, then we define (a_2, t_1^B) as a problematic pair. A published dataset T is considered as $safe$ if it has no problematic pairs in it. In this privacy problem, for a given dataset T of trajectories, a probability threshold P_{br} and a set of adversaries, the goal of a data publisher is to construct a safe dataset corresponding to T with minimum information loss.

IV. Existing Approaches

In this section, we discuss relevant existing work. Terrovitis et al. [1] first proposed a global suppression algorithm for anonymizing trajectory datasets. They used a greedy approach that iteratively scans the whole dataset and suppresses locations on each iteration until the privacy constraints are met. The algorithm first identifies the projections of each adversary which cause privacy problem by scanning the whole dataset once. Then it iteratively unifies a large projection by a small projection of the same adversary to reduce the number of problems in the original dataset. A large projection t_R^A is suppressed by a small projection t_r^A if the suppression generates maximum utility gain. In each iteration, the algorithm chooses a pair of projections that gives maximum utility gain upon suppression. The utility gain calculation ensures more anonymity gain and less information loss. When a large projection t_R^A is suppressed by a small projection t_r^A, the anonymity gain δ is defined as

$$\delta = \frac{N - N'}{N}. \tag{2}$$

Here, N is the total number of problems before the suppression and N' refers the total number of problems in the dataset after suppression. The information loss of a trajectory t caused by such suppression is defined as

$$ploss(t, t') = 1 - \frac{|t'|(|t'| - 1)}{|t|(|t| - 1)}. \tag{3}$$

Here, t is the trajectory before suppression and t' is the trajectory after performing the suppression. The lower $ploss$

values specify less information loss. The total utility gain for suppression from t_R^A to t_r^A is defined as

$$u_{gain}(t_R^A, t_r^A) = \delta \times \frac{1}{\sum_{t \in S} ploss(t, t')}, \quad (4)$$

where t_R^A and t_r^A are two projections of adversary A, and S is a set of trajectories which are affected due to the suppression operation. After each suppression, the algorithm re-assesses the number of problems from the suppressed dataset and repeats the same process again. In each iteration, the algorithm scans the whole dataset to identify the problematic projections of each attacker. Moreover, it needs to scan the whole dataset to identify the pair of projections that maximizes the utility gain. For a large trajectory dataset, the process is computationally expensive in every iteration.

V. Proposed Algorithm

We propose a global suppression algorithm for trajectory data anonymization using an efficient data structure named Projection Support Tree (PST). Our algorithm (Global Suppression - PST) is a modified version of the global suppression algorithm proposed in [1]. We use the projection support tree data structure on the global suppression algorithm [1] to reduce extensive search complexity. Formation of the projection support tree and the proposed global suppression algorithm are discussed in the following subsections.

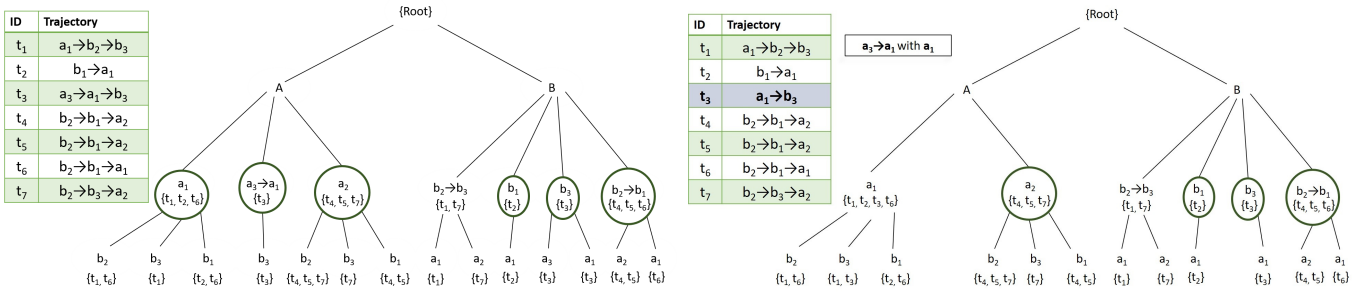
A. Projection Support Tree (PST)

We construct the projection support tree from the trajectory dataset T. The tree contains all possible projections along with the problematic pairs with respect to the adversaries. Initially, an empty node is added to the tree (at level 0 of the tree) which is considered as the root of the PST. Each adversary $A \in \mathcal{A}$ is added as a child of the root. For each adversary $A \in \mathcal{A}$, Algorithm 1 scans each trajectory $t \in T$ and retrieves the projection $\pi_A(t)$. Then the projection $\pi_A(t)$ was added to the child node of A (if $\pi_A(t)$ is not empty and it is not added previously) (Lines 7-11 in Algorithm 1). Each node $\pi_A(t)$ at the second level of the tree has an array containing a set of trajectories which have the projection $\pi_A(t)$. Therefore, we add the trajectory t into the array of the node $\pi_A(t)$. The node a_1 at the second level of the tree in Figure 1a has an array $\{t_1, t_2, t_6\}$ which indicates that t_1, t_2 and t_6 trajectories have projection a_1 for the adversary A. Each node at the second level of the tree also has a boolean flag which defines whether the projection of the node carries a problematic pair in the dataset. Initially, the flags of all the nodes are set to false. In Figure 1, the flagged nodes are represented with circles. After adding the trajectory t into the array of the node $\pi_A(t)$, algorithm 1 scans each location $\lambda \in t \setminus \{\pi_A(t)\}$ on this trajectory. Each λ is added as a child node of the parent node $\pi_A(t)$ if it is not added before (line 12-17 in algorithm 1). These child nodes also have an array which contains the list of trajectories which have the projection $\pi_A(t)$ and the location λ. Therefore, we add t into the array of the child node λ. In Figure 1a, the node $a_3 \rightarrow a_1$ at the second level of the tree has a children b_3 which has an array $\{t_3\}$. This indicates the trajectory t_3 has projection $a_3 \rightarrow a_1$ and the location b_3. In the above away, we can create the whole projection support tree for all the trajectories with all the adversaries.

After creating the projection support tree, we can update the tree based on performing a suppression operation in the dataset. In the update operation (Algorithm 2), we want to suppress a projection mt_R with another projection mt_r. Note that, both mt_R and mt_r belong to the same adversary. At the beginning, we search the two nodes with projection mt_R and mt_r at the second level of the tree. Then we find the trajectories that are present in the array of mt_R but are not present in the array of mt_r. We add these trajectories to the array of the node mt_r. Moreover, all the child nodes of mt_R are added to the child node of mt_r. If the node mt_r already has such children, then we only update the array of those children. In Figure 1b, we suppress the projection $a_3 \rightarrow a_1$ with a_1. Therefore, we add trajectory t_3 to the array of node a_1 and we update the trajectory list of a_1's child node b_3 by adding t_3 into the list. Afterward, we check whether any of a_1's child c and any node c' at the second level of the tree have the same projection. If we find such a node, then we update the trajectory list of c''s child (Lines 7-10 in Algorithm 2). In Figure 1b, b_3 is the child of $a_3 \rightarrow a_1$ before suppression. However, b_3 has also appeared in a node at the second level of the tree. Since $a_3 \rightarrow a_1$ was only in trajectory t_3, we remove t_3 from b_3's child node a_3. As the trajectory list of the node a_3 becomes empty, we remove the node a_3 from the tree. After finishing all the updates, we remove the node mt_R from the second level of the tree. In this way, we update the whole tree after performing each suppression operation.

Algorithm 1 PST - Creation

```
1:  procedure CreatePST(T, A)
2:      tree = null
3:      Let root be a new node and insert root to the tree
4:      for each A ∈ A do
5:          insert A to the root of tree
6:          for each t ∈ T and π_A(t) ≠ φ do
7:              if π_A(t) ∉ A's Children then
8:                  insert π_A(t) as A's child and set π_A(t).flag = false
9:                  add t into π_A(t).trajectory
10:             else
11:                 add t into π_A(t).trajectory
12:             for each λ ∈ t \ {π_A(t)} do
13:                 if λ ∉ π_A(t)'s child then
14:                     insert λ as π_A(t)'s child
15:                     add t into λ.trajectory
16:                 else
17:                     add t into λ.trajectory
18:     return tree
```



(a) PST Creation

(b) PST Update after projection $a_3 \rightarrow a_1$ is suppressed by a_1

Figure 1: Projection Support Tree (PST) creation and update (considering the dataset given in the figure and P_{br} is 0.5)

Algorithm 2 PST - Update

```
1:  procedure UpdatePST(mt_R, mt_r, tree)
2:      Let n be the node at 2^nd-level of tree having projection mt_R
3:      if mt_r ≠ φ then
4:          Let n' be the node at 2^nd-level of tree having projection mt_r
5:          Update n'.trajectory
6:          Update the trajectory set of all n''s child
7:      for each child c of n do
8:          for each node c' in the 2^nd-level of tree do
9:              if c and c' have same projection then
10:                 Update c''s child
11:     Delete node n from tree
12:     return tree
```

B. Global Suppression using PST

In this subsection, we modify the global suppression algorithm [1] using the projection support tree data structure. Identifying the number of threats from the projection support tree is more simple and faster than the algorithm proposed in [1]. In Algorithm 3, we calculate the number of threats from the projection support tree. We scan all nodes at the second level of the tree. Initially, we set the flag of all these nodes to false. The set of trajectories in the array of these nodes is their support set. For each node n at the second level, we check whether n has a problematic pair or not for any of its children c (Line 6 in Algorithm 3). We set the flag of n to true if it is problematic for c and increase the number of problems by the number of trajectories in c's array. In Figure 1a, node a_1 is problematic for its children b_2 and b_1 considering P_{br} is 0.5.

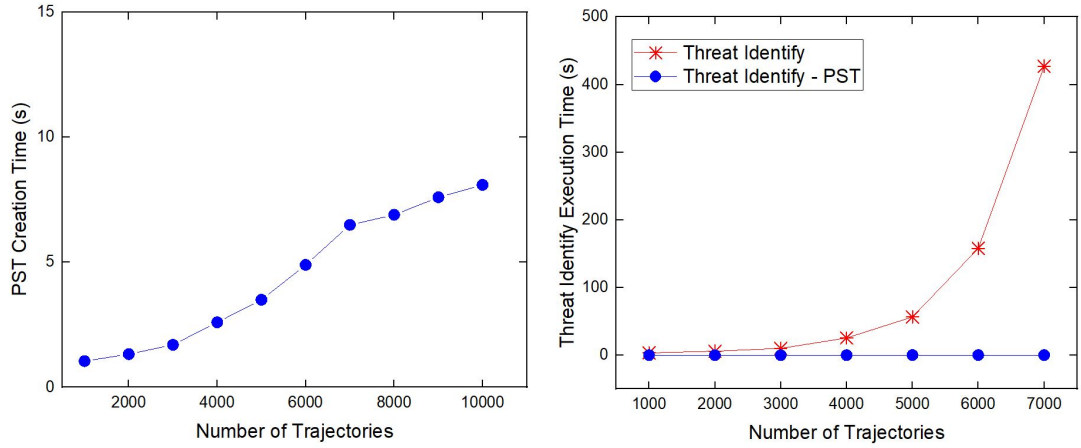
Algorithm 3 Threat Identify - PST

```
1:  procedure ThreatPST(tree, P_br)
2:      N = 0
3:      set the flag of all the nodes in the 2^nd-level of tree to false
4:      for each child n of tree → root do
5:          for each child c of n do
6:              if |c.trajectory| / |n.trajectory| > P_br then
7:                  n.flag = true
8:                  N = N + |c.trajectory|
9:      return N
```

Algorithm 4 Global Suppression - PST

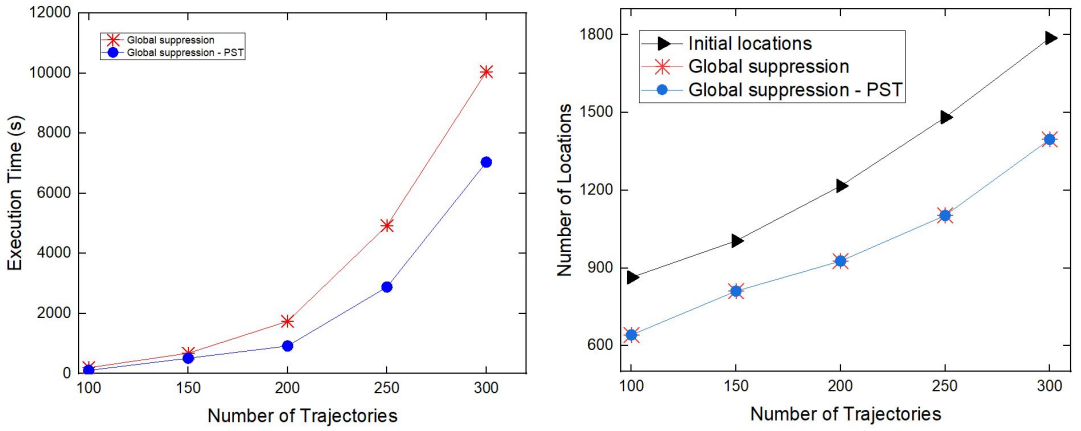
```
1:  procedure GlobalSuppressionPST(T, A, P_br)
2:      tree = CreatePST(T, A)
3:      N = ThreatPST(tree)
4:      Q = φ                          ▷ Queue for storing suppression pairs
5:      while N > 0 do
6:          for each node n in the 1^st-level of tree do
7:              for each pair of (t_R, t_r) s.t. t_R ∈ n's children and (t_r ∈
    n's children or t_r = φ) and t_R ≠ t_r do
8:                  if t_R.flag or t_r.flag is true and t_r ⊂ t_R then
9:                      Calculate U_gain(t_R, t_r)
10:         Let mt_r and mt_R be the projections with maximum U_gain
11:         tree = UpdatePST(mt_R, mt_r, tree)
12:         Insert pair (mt_R, mt_r) to queue Q
13:         N = ThreatPST(tree)
14:     while Q is not empty do
15:         (t_R, t_r) = Q.dequeue()
16:         Update T for suppressing t_R by t_r in all trajectories
17:     return T
```

In Global Suppression - PST (Algorithm 4), at first we create the projection support tree of the dataset using Algorithm 1. After creating the tree, we use it to calculate the number of problems in the dataset by calling Algorithm 3. Algorithm 4 iteratively suppresses trajectories until the total number of problems in the dataset becomes 0. A projection t_R can be suppressed by another projection t_r only if $|t_R| > |t_r|$, t_r is a sub-trajectory of t_R and one of t_R or t_r is problematic. In iteration, we find a projection pair mt_r and mt_R which produces maximum utility gain (using equation 4). In our algorithm, finding such a projection pair is computationally less expensive as we are searching projections at the second level of the projection support tree. From the nodes at the second level of the tree, we know whether a projection is problematic by seeing its flag. Since the projections in the tree are stored as children of their respectful adversaries, we do not need to search through all possible projections. After finding mt_R and mt_r, we update the tree using Algorithm 2. This update operation ensures that the projection mt_R is suppressed by the mt_r in the tree. We add the projection pair (mt_R, mt_r) to a queue Q and calculate the number of problems from the updated tree. If the number of problems is greater than 0, we continue the same procedure until it



(a) PST Creation

(b) Threat identification time



(c) Total execution time

(d) Locations after anonymization

Figure 2: Performance comparison between Global Suppression and Global Suppression - PST for $BMS-POS$ dataset

becomes 0. When the number of problems reduces to 0, the dataset becomes safe. Finally, we deque each of the pair from Q and perform the suppression to the actual dataset for that pair (Lines 14-16 of Algorithm 4).

C. Complexity Analysis

We consider k be the total number of suppressions required to eliminate all problematic pairs. The running time of the projection support tree creation algorithm is $O(|\mathcal{A}||T||\mathcal{L}|)$. We assume $|\pi|$ to be the total number of projections in the dataset with respect to all the adversaries. However, the number of projections is reduced in each step due to the suppression. The update operation in the projection support tree has a running time of $O(|\pi||\mathcal{L}|)$. The running time of the threat identification from the projection support tree is $O(|\pi||\mathcal{L}|)$. Therefore, the total running time of our proposed global suppression algorithm is $O(k|\pi||\mathcal{L}|)$. Therefore, the total running time of existing global suppression algorithm [1] is $O(k|\mathcal{A}||T||\mathcal{L}|)$. In the worst the case, the total number of projections would be equal to $|\mathcal{A}| \times |T|$. Nevertheless, this worst case assumption is not practical considering to the real world data sets.

VI. Performance Evaluation

In order to evaluate the performance of our proposed data structure, we implemented all algorithms in Java and tested them on an Intel Core i5 at 1.60 GHz with 8 GB of RAM. We used a real-world dataset $BMS-POS$ [17] in our experiment to have a realistic setting. The dataset consists of a transaction log of sales of an electronic retailer. Each record of the dataset is a trajectory of a customer in different stores. We distribute the stores into 5 different adversaries. The threshold probability P_{br} value is 0.5 in all the experiments. We compared the performance of our algorithm with the global suppression algorithm proposed in [1].

The execution time of projection support tree creation is shown in Figure 2a. The execution time increases with the increasing number of trajectories. However, for 10000

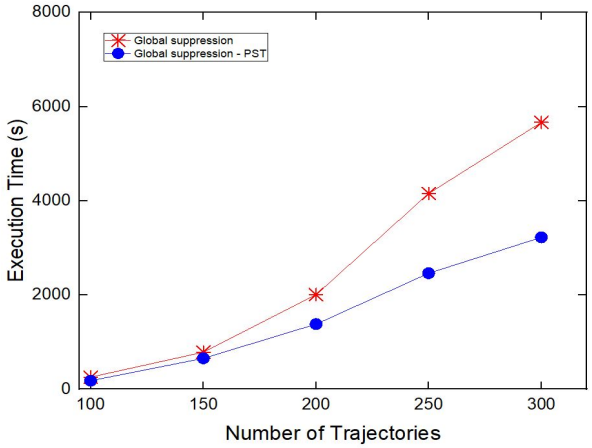


Figure 3: Execution time of the algorithms on a varying the number of trajectories on a random dataset

trajectories the execution time does not exceed 10 seconds. We create the projection support tree only once at the beginning of Algorithm 4. Therefore, it has less effect on the computational complexity of the algorithm. In Figure 2b, the comparison of execution time between threat identification in [1] and threat identification using PST is depicted. The execution time of threat identification in [1] increases rapidly when we increase the number of trajectories. For 7000 trajectories, the algorithm takes almost 450 second to compute the number of problems. The results indicate that the threat identification using PST does not take more than 1 second for up to 7000 trajectories. This faster execution time makes our Algorithm 4 much faster than the global suppression algorithm proposed in [1] which is depicted in Figure 2c. For 300 trajectories the execution time of our algorithm is significantly less than the algorithm in [1]. We also run the experiment for a random synthetic dataset where the average lengths of all trajectories is 4 to 7 and the number of unique locations is 100. For this randomly generated dataset, our proposed algorithm also outperforms the existing algorithm (Figure 3).

In order to check the correctness of our algorithm, we calculated the number of distinct location in the anonymized dataset for Global Suppression - PST and Global Suppression algorithm. Figure 2d shows that the number of distinct location in anonymized dataset is identical for both the algorithms. The above discussion suggests that the Global Suppression - PST outperforms the state-of-the-art approach in terms of the running time of the algorithm.

VII. Conclusion

In this paper, we have studied location trajectory data anonymization techniques. Here, the trajectory data is collected from a customer's credit card transaction history. An adversary can have partial knowledge of the published dataset and identify some individuals' complete trajectories. A global suppression algorithm is presented in the literature which is computationally expensive for large trajectory datasets. We propose a projection support tree data structure which makes the global suppression algorithm much faster. Moreover, the data structure has an efficient update operation that produces the correct solution. In future, we would like to explore applying this data structure to local suppression and splitting algorithms for the trajectory data anonymization problem.

Acknowledgment

The research is supported in part by the NSERC Discovery Grants (RGPIN-2015-04147).

References

[1] M. Terrovitis, G. Poulis, N. Mamoulis, and S. Skiadopoulos, "Local suppression and splitting techniques for privacy preserving publication of trajectories," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 7, pp. 1466–1479, 2017.

[2] J. Gudmundsson, P. Laube, and T. Wolle, "Computational movement analysis," in *Springer handbook of geographic information*, pp. 423–438, Springer, 2011.

[3] M. Baza, N. Lasla, M. Mahmoud, G. Srivastava, and M. Abdallah, "B-ride: Ride sharing with privacy-preservation, trust and fair payment atop public blockchain," *IEEE Transactions on Network Science and Engineering*, 2019.

[4] C.-Y. Lin, Y.-C. Wang, W.-T. Fu, Y.-S. Chen, K.-C. Chien, and B.-Y. Lin, "Efficiently preserving privacy on large trajectory datasets," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pp. 358–364, IEEE, 2018.

[5] G. Ghinita, "Privacy for location-based services," *Synthesis Lectures on Information Security, Privacy, & Trust*, vol. 4, no. 1, pp. 1–85, 2013.

[6] R. Trujillo-Rasua and J. Domingo-Ferrer, "On the privacy offered by (k, δ)-anonymity," *Information Systems*, vol. 38, no. 4, pp. 491–494, 2013.

[7] O. Abul, F. Bonchi, and M. Nanni, "Anonymization of moving objects databases by clustering and perturbation," *Information Systems*, vol. 35, no. 8, pp. 884–910, 2010.

[8] C. Clifton and T. Tassa, "On syntactic anonymity and differential privacy," in *2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*, pp. 88–93, IEEE, 2013.

[9] K. Jiang, D. Shao, S. Bressan, T. Kister, and K.-L. Tan, "Publishing trajectories with differential privacy guarantees," in *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, p. 12, ACM, 2013.

[10] X. He, G. Cormode, A. Machanavajjhala, C. M. Procopiuc, and D. Srivastava, "Dpt: differentially private trajectory synthesis using hierarchical reference systems," *Proceedings of the VLDB Endowment*, vol. 8, no. 11, pp. 1154–1165, 2015.

[11] L. Bonomi and L. Xiong, "A two-phase algorithm for mining sequential patterns with differential privacy," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pp. 269–278, ACM, 2013.

[12] T. Hikita and R. S. Yamaguchi, "Preliminary study about advantageous trajectory anonymization methods based on population," in *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, pp. 492–495, IEEE, 2018.

[13] M. E. Nergiz, M. Atzori, and Y. Saygin, "Towards trajectory anonymization: a generalization-based approach," in *Proceedings of the SIGSPATIAL ACM GIS 2008 International Workshop on Security and Privacy in GIS and LBS*, pp. 52–61, ACM, 2008.

[14] M. Terrovitis, N. Mamoulis, and P. Kalnis, "Privacy-preserving anonymization of set-valued data," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 115–125, 2008.

[15] M. Terrovitis, N. Mamoulis, and P. Kalnis, "Local and global recoding methods for anonymizing set-valued data," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 20, no. 1, pp. 83–106, 2011.

[16] R. Chen, B. C. Fung, N. Mohammed, B. C. Desai, and K. Wang, "Privacy-preserving trajectory data publishing by local suppression," *Information Sciences*, vol. 231, pp. 83–97, 2013.

[17] Z. Zheng, R. Kohavi, and L. Mason, "Real world performance of association rule algorithms," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 401–406, ACM, 2001.