Competitive Privacy: Secure Analysis on Integrated Sequence Data

Raymond Chi-Wing Wong
Hong Kong University of Science and Technology

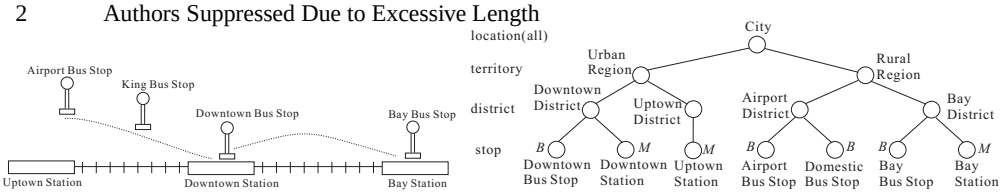
Eric Lo
Hong Kong Polytechnic University

Abstract. Sequence data analysis has been extensively studied in the literature. However, most previous work focuses on analyzing sequence data from a single source or party. In many applications such as logistics and network traffic analysis, sequence data comes from more than one source or party. When multiple autonomous organizations collaborate and integrate their sequence data to perform analysis, sensitive business information of individual parties can be easily leaked to the other parties. In this paper, we propose the notion of competitive privacy to model the privacy that should be protected when carrying out data analysis on integrated sequence data. We propose a query restriction algorithm that can reject malicious queries with low auditing overhead. Experimental results show that our proposed method guarantees the protection of competitive privacy with only a significantly small portion of queries being restricted.

1 Introduction

Sequence data analysis has been studied extensively in the literature [4, 6, 2]. Most previous work focuses on analyzing sequence data collected from a single source or party. However, in applications such as logistics and network traffic analysis, some autonomous enterprises may want to integrate their sequence data in order to carry out joint data analysis. As a motivating example, consider the collaboration between a bus company B and a metro company M in a city that has implemented RFID-based electronic transportation payment systems (e.g., Washington DC's SmarTrip system). Each passenger has an RFID-card that can be used as a form of e-money for the fare of various transportations. Each transportation company participates in the e-transport network records a huge volume of passenger transactions every day. In this example, we can view each passenger traveling history as a data *sequence*. In Figure 1a, if a passenger traveled from "Airport Bus Stop" to "Downtown Bus Stop" by bus, transferred from "Downtown Bus Stop" to "Downtown Station" (via a transfer terminal in "Downtown") and finally traveled from "Downtown Station" to "Uptown Station" by metro, her traveling history can be represented as a data sequence ("Airport Bus Stop", "Downtown Bus Stop", "Downtown Station", "Uptown Station").

Suppose that B and M collaborate and offer discounts to passengers who traveled from the airport to uptown using a combination of bus and metro (transited at Downtown). One interesting query is to ask the number of passengers who traveled from "Airport Bus Stop" to "Uptown Station" via the transfer terminal in "Downtown". Furthermore, during data analysis, queries are often refined to different abstraction levels by the data analysts interactively. For example, if a concept hierarchy is defined for stations/stops like the one in Figure 1b, then the above query may be "rolled-up" by the user to ask for the number of passengers who traveled from "Airport District" to



(a) A transportation network (b) A concept hierarchy

Fig. 1. Motivating Example

"Uptown District" via the transfer terminal in "Downtown". All these operations can be handled by sequence analytical systems such as [2] and [4] efficiently. One way to evaluate the analytical queries above is to have bus and metro to *integrate* their passenger data, which are originally *owned* and stored separately. Let D_M be the data owned by M and D_B be the data owned by B. D_M and D_B are integrated to form a new dataset D_I. In practice, however, both M and B actually do not want to disclose their data to their competitors, if possible. For instance, assume that there are two services operated by M and B separately from "Downtown District" to "Bay District". Specifically, M operates a service s_M from "Downtown Station" to "Bay Station" while B operates a service s_B from "Downtown Bus Stop" to "Bay Bus Stop". If passengers want to travel from "Downtown District" to "Bay District", they may choose either s_M or s_B. Thus, these two services s_M and s_B are *competitive*. Suppose that M poses a query and observes that the total number of passengers using service s_B (operated by B) is extremely large compared with its own service s_M. M may then offer discounts to customers who use its service s_M in order to attract the customers originally using service s_B. It is easy to see that, once there are discounts for service s_M, the original service s_B operated by B is definitely affected. Thus, the statistical information about the total number of passengers using service s_B can be regarded as the "competitive privacy" of party B and that should get protected during data analysis.

The objective of this paper is to support data analysis, in particular, OLAP, on an integrated sequence data set without compromising competitive privacy. Informally, let $Q(s_B, f)$ be an aggregate sequence query [2] that specifies an aggregate function f on all the sequences in the integrated data set D_I that match s_B and the data values in s_B are all *owned by* (or *originated from*) party B (formal definitions are given in Section 2), we say that there is a breach of *competitive privacy* if given a real number e, other parties (except B) can infer a value $\widetilde{f}$ such that $|\widetilde{f} - f(s_B)| \leq e$, where $f(s_B)$ denotes the answer of query Q. In this paper, we present a query restriction strategy to support data analysis on an integrated sequence data set without breaching the competitive privacy of any party. The strategy rejects a query Q if its answer can lead to a breach of competitive privacy. Existing query restriction strategies like [1], [5] and [3] focus on the protection of individual privacy or data privacy on a relational data set owned by a single party. The query restriction strategy in this paper focuses on the protection of competitive privacy on a sequence data set integrated from multiple autonomous parties.

2 Preliminary

We are given a set $\mathcal{V}$ of values that are associated with a concept hierarchy. Figure 1b shows a concept hierarchy. Nodes at the leaf level correspond to the values recorded in the data. A node N is said to be *ground* if it is at the leaf level or *non-ground* if it is not.

Each value in $\mathcal{V}$ corresponds to a node in the concept hierarchy. Without ambiguity, in the following, the terms "nodes" and "values" are used interchangeably. Each leaf

node N is associated with an *ownership*, denoted by $N.T$. For example, since "Downtown Station" and "Uptown Station" are values originated from the metro company M's data, the *ownership* of these nodes are M. In Figure 1b, the ownership of a leaf node is next to itself. The non-leaf nodes such as "Downtown District" and "Urban Region" are used for data analysis and they do not have any ownership.

Suppose there are m datasets $\mathcal{D} = \{D_1, D_2, ..., D_m\}$ owned by m parties $P_1, P_2, ..., P_m$, respectively. Each data set contains a number of sequences. A *sequence* s is represented in the form of $(N_1, N_2, ..., N_k)$ where N_l is a ground or non-ground value in $\mathcal{V}$ for $l \in [1, k]$. We say that this sequence s is of *length* k. Implicitly, each value N_i in s is associated with a timestamp, denoted by $N_i.ts$, such that if $i < j$, $N_i.ts < N_j.ts$ for any $i, j \in [1, k]$. Each sequence s in dataset D_i is associated with a unique identifier, denoted by $s.id$ (e.g., the card id of an RFID card).

An integrated dataset D_I can be obtained by integrating the set of databases $\mathcal{D}$ according to the timestamp of the values. Specifically, let $\mathcal{C}$ be the set of sequence identifiers in $\mathcal{D}$. For each $x \in \mathcal{C}$, we obtain a set $\mathcal{S}$ of sequences from all datasets in $\mathcal{D}$ such that $\mathcal{S} = \{s \in \mathcal{D} | s.id = x\}$. Let $\mathcal{N}$ be the multi-set containing the values of all sequences in $\mathcal{S}$. We generate a new sequence s' of length $|\mathcal{N}|$ in the form of $(N_1, N_2, ...N_{|\mathcal{N}|})$, such that if $i < j$, $N_i.ts < N_j.ts$ for any $i, j \in [1, |\mathcal{N}|]$. The new sequence s' will be inserted into the integrated dataset D_I.

In this paper, we focus on aggregate sequence queries [2]. If such a query $Q(s, f)$, or simply Q if the context is clear, is posed on a sequence data set D_I, it applies an aggregate function f on all the sequences in D_I that MATCH s, and returns a scalar value, denoted as $f(s)$, to a user. We remark that MATCH can be any pattern matching function. For example, it can be a sub-string matching function (i.e., if s is a sub-string of a sequence s' in D_I, MATCH returns `true`) or a sub-sequence matching function (i.e., if s is a sub-sequence of s' in D_I, MATCH returns `true`). The technique in this paper is applicable to all kinds of aggregate sequence queries discussed in [2]. Nonetheless, for the sake of illustration, the following discussion mainly centers around the `COUNT` aggregation function and the sub-string matching function. Therefore in the following, unless stated otherwise, we assume a query $Q(s, f)$ on D_I means that for each sequence in D_I which contains s as substring (despite the number of occurrences of s in a sequence) increments the value of $f(s)$ by one. A query $Q(s, f)$ is of *length* k if the length of sequence s specified in Q is k. We denote that by $|Q|$. As the data is actually integrated from multiple parties, we assume that all queries are of length at least two.

As in traditional OLAP environments, users may interactively refine their queries. For instance, a user (of party P_i) may first issue a query to obtain the number of customers who traveled from "Airport Bus Stop" to "Uptown Bus Stop" and then refine her query Q by a "pattern roll-up" operation [2] in order to obtain the number of passengers who traveled from "Airport District" to "Uptown District". These concepts can be formalized as follows.

Assume party P_i issues her first query Q_1 at time $t = 1$, second query Q_2 at time $t = 2$ and so on. Let $\mathcal{K}_{P_i}(t)$ be the knowledge of party P_i at time t. Thus, the initial knowledge of party P_i before she issues any query on the integrated data set D_I, denoted as $\mathcal{K}_{P_i}(1^-)$, contains all aggregate values $f(s)$ for all s in D_i. Further, let t^- and t^+ be the time *immediately* before and after time t, i.e., t^- is any time between time

$t-1$ and time t, and t^+ be any time between time t and time $t+1$. Thus, for any $t > 1$, $\mathcal{K}_{P_i}(t^-) = \mathcal{K}_{P_i}((t-1)^+)$. After party P_i issues a query Q_t at time t, if Q_t is not rejected, P_i's knowledge $\mathcal{K}_{P_i}(t^+)$ is immediately updated to $\mathcal{K}_{P_i}(t^-) \cup \{f(s)\}$ (where $f(s)$ is the answer of Q_t); otherwise, P_i's knowledge $\mathcal{K}_{P_i}(t^+)$ remains as $\mathcal{K}_{P_i}(t^-)$.

3 Competitive Privacy

In this section, we present the concept of *competitve privacy*, which is the key element that we should consider when supporting data analysis on a sequence data set that is integrated from multiple autonomous parties. We assume that the integrated data set D_I is located at trusted party T and the involved parties send their queries to T. The RFID transport payment company can be regarded as the trusted parties for the motivating example. Although a trusted party is involved, the privacy issue has not been resolved yet. Specifically, in the following, we are going that formalize the concept of competitive privacy and show that if a party P can pose any queries without any restriction, that will breach competitive privacy of some party. Let us begin with the definition of *conflicting node set*. Given a ground node N, the *conflicting node set* of N, denoted by $C(N)$, is a set of ground nodes such that for each node $N' \in C(N)$, $N'.T \neq N.T$. The conflicting node set is *specified* or *given* by the multiple autonomous parties and the trusted party.

In our running example, if the metro company offers a service from "Downtown Station" to "Bay Station" and the bus company offers a service from "Downtown Bus Stop" to "Bay Bus Stop", then the manager of the metro may specify that the conflicting node set of "Downtown Station" as {"Downtown Bus Stop"}. Similarly, the manager is likely to specify that the conflicting node set of "Bay Station" as {"Bay Bus Stop"}. We remark that we assume the notion of conflicting node set is symmetric in this paper. As a result, if C("Downtown Station")={"Downtown Bus Stop"}, then C("Downtown Bus Stop'")={"Downtown Station"}.

Given a sequence s_i in D_i in the form of (N_p, N_q) and another sequence s_j of D_j in the form of (N_r, N_s), s_j is a *competitive sequence* of s_i, if $N_r \in C(N_p)$ and $N_s \in C(N_q)$. The set of competitive sequences of s_i is denoted by $C(s_i)$. For example, let s_M=("Downtown Station", "Bay Station") in D_M and s_B=("Downtown Bus Stop", "Bay Bus Stop") in D_B. Following the example above, as "Downtown Station" $\in C$("Downtown Bus Stop") and "Bay Station" $\in C$("Bay Bus Stop"), s_M is a competitive sequence of s_B (and vice versa because of the symmetric property). Note that instead of asking the managers (which are the target users of OLAP systems) to specify the (query) views that needed to be protected as in [3], we intentionally introduce the notion of conflicting node such that it is more non-technical people friendly. For example, rather than directly specifying s_M and s_B as competitive sequences, it would be more intuitive for those business people, say, the operation manager of party M to directly specify "Downtown Station" and "Downtown Bus Stop" as "conflicting". Nonetheless, of course, it is also possible for users to directly specify competitive sequences as well. Now, we can define the competitive privacy of a party P as follows.

Definition 1 (Competitive Privacy). *The competitive privacy $\mathcal{CP}$ of a party P_i is defined as the statistical information of all competitive sequences in D_i, i.e., $\mathcal{CP}$ = $\{f(s) | \forall s \in D_i$ and there exists $s' \in D_j$ such that $j \neq i$ and $s' \in C(s)\}$.* □

Similar to what we discussed in Section 1, the statistical information of each competitive sequences, namely $\mathcal{CP}$, are regarded as the "competitive privacy" of a party.

Thus, without any query restriction, a party can *directly* obtain the statistical information of the competitive sequence of the other party easily and can do something bad to the other party.

We now show that party M can infer a value for $\texttt{COUNT}(s_B)$, through *query inferences*, even though it only obtains the statistical information other than the value of $\texttt{COUNT}(s_B)$.

Example 1 (Query Inferences). In our motivating example, both the bus (party B) and the metro (party M) offer services from Downtown district to Bay district. Assume that each of the parties provide only one service from Downtown district to Bay district.

Initially, $\mathcal{K}_M(1^-) = \{\texttt{COUNT}(s_M)\}$. Suppose at time 1, M issues a query $Q_1(\hat{s}, \texttt{COUNT})$, where $\hat{s}$=("Downtown District", "Bay District") (where the concept hierarchy is the one in Figure 1b). Without any query restriction, Q_1 can be posed on D_I and thus the knowledge of M can be updated to $\mathcal{K}_M(1^+) = \{\texttt{COUNT}(s_M), \texttt{COUNT}(\hat{s})\}$. Assume $\texttt{COUNT}(s_M) = 10,000$ and $\texttt{COUNT}(\hat{s}) = 90,000$, Party M can infer a value for $f(s_B)$ as $\texttt{COUNT}(\hat{s}) - \texttt{COUNT}(s_M) = 80,000$ □

Definition 2 (Competitive Privacy Breach). *Given two competitive sequences s_i and s_j obtained from D_i and D_j respectively. At time t, we say that there is a* competitive privacy breach *with respect to party P_j by party P_i if, given a real number e, P_i can infer a value $\widetilde{f}(s_j|\mathcal{K}_{P_i}(t^-))$ for $f(s_j)$ such that $|\widetilde{f}(s_j|\mathcal{K}_{P_i}(t^-)) - f(s_j)| \leq e$ based on knowledge $\mathcal{K}_{P_i}(t^-)$.* □

4 Query Restriction

In this section, we will give a high-level description of the proposed algorithm called CCF (conservative competition-free) to avoid any competitive privacy breach. Details of this algorithm can be found in [7]. Intuitively, we reject some queries which may breach competitive privacy. Consider a query Q which has sequence s. We reject query Q if one of the following two conditions holds. *Condition 1:* There exists a competitive sequence which is a sub-sequence of s. *Condition 2:* There exists a *generalized version* of competitive sequence which is a sub-sequence of s. We say that sequence $s_i = (N_1, N_2, ..., N_l)$ is a *generalized version* of another sequence $s_j = (M_1, M_2, ..., M_l)$ if N_x is equal to M_x or is an ancestor node of M_x (in the concept hierarchy) for all $x \in [1, l]$. In [7], we prove that our query restriction algorithm can avoid any competitive privacy breach.

5 Empirical Study

We have conducted extensive experiments on a Pentium IV 2.4GHz PC with 1GB memory, on a Linux platform. The programs were implemented in C++. We evaluated our algorithm, CCF, on both synthetic and real datasets, in terms of four measurements: (1) *average auditing time*, (2) *ratio of restricted queries*, and (3) *storage*. The average auditing time corresponds to the average time to check whether a query is rejected by our proposed algorithm CCF. The ratio of restricted queries is equal to the total number of restricted queries by CCF over the total number of issued random queries. The storage corresponds to the memory usage to hold all competitive sequences of all parties, namely $\mathcal{CS}$. All experiments were conducted 100 times and we took the average for the results. In our experiments, we generate 10,000 batches of queries. Each batch contains

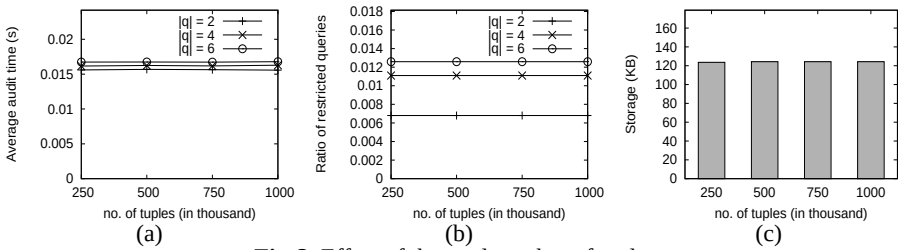


Fig. 2. Effect of the total number of tuples

20 queries. We randomly generate a query Q_1 with sequence $s = (N_1, N_2, .., N_{|Q_1|})$ as follows. For each N_i where $i \in [1, |Q_1|]$, we randomly select a value in the concept hierarchy. Then, we refine query Q_1 and generate another new query Q_2. We adopt the refinement operations from [2]: Append, De-tail, Pre-pend, De-head, Pattern-roll-up and Pattern-drill-down. We randomly select one of the operations and generate query Q_2. Similarly, we repeatedly generate query Q_i from Q_{i-1} until $i = 20$.

Synthetic Dataset: The synthetic dataset is generated by a dataset generator. This generator creates sequences with 4 parameters, namely n, p, c and l, where n is the total number of (integrated) sequences, p is the total number of parties, c is the percentage of ground competitive sequences in each party's dataset, and l is the average length of the data sequences. The data sequence is generated as the same way as [2] and we randomly assign c% of sequences as competitive. We generate a generalization hierarchy of height 3. We partition the ground nodes into different groups such that different ground nodes, say v and v', where $v \in C(v')$ (or $v' \in C(v)$) forms the same group. For each group of ground nodes, we create an internal node N. Finally, we create a root node N' such that the parent of all internal nodes constructed is N'. Thus, the final hierarchy has height equal to 3. The default values of n (num. of tuples), p (num. of parties), c (the ratio of ground competitive sequences) and l (average sequence length) are 500K, 4, 0.05, and 20, respectively. In the experiments, we study the effect of the total number of tuples and the length of the query.

In Figure 2(a), the average audit time remains nearly unchanged when the dataset size changes. The auditing time of our proposed algorithm mainly depends on the size of $\mathcal{CS}$ (Details can be found in [7]). Since the size of $\mathcal{CS}$ is fixed (Figure 2(c)), the change in the dataset size does not affect the auditing time too much. Besides, we can observe that the average auditing time increases with $|Q|$, the length of the query. Figure 2(b) shows that the number of restricted queries is nearly the same with different dataset size. Similarly, since the percentage of competitive sequences remains unchanged when the dataset size changes, the ratio of restricted queries also remains unchanged. When $|Q|$ increases, it is trivial that the ratio increases. Figure 2(c) shows the storage of algorithm CCF keeps unchanged when the dataset size increases. This is because the storage for set $\mathcal{CS}$ is independent of the dataset size.

Real Dataset: The real dataset is obtained from a local transportation organization called MTR in Hong Kong. It consists of passenger transactions of 5-working-day, all recorded by an RFID-based electronic payment system. The passenger transactions are consolidated from 4 different in-city railway lanes. Each lane corresponds to a party. There are 63 stations in total and 6 of them are transfer terminals. In particular, 5

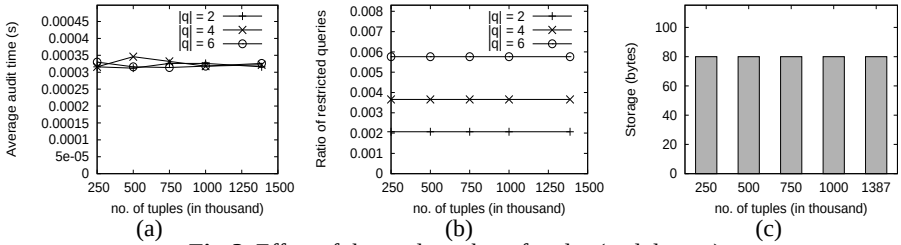


Fig. 3. Effect of the total number of tuples (real dataset)

transfer terminals allow passengers to switch to another lane, and 1 transfer terminal is a hub that allow passengers to switch to two other lanes. An example record is like (N_1, N_2, N_3, N_4), which denotes that there was a passenger entered the railway network at station N_1, got off at station N_2, transferred to another lane at station N_3 and finally left the railway network at station N_4. All pairs of transfer terminals as defined as conflicting. That is, in this example, $C(N_2) = \{N_3\}$. All together we have 1,387,831 sequence records. The average sequence length of a record is 2.9 stations. According to the locations of stations, we divide the stations into different regions such that there are 63 leaf values and 31 non-leaf values in the concept hierarchy of height 4.

We carry out experiments that are similar to the results for synthetic datasets. Figure 3 shows that the experimental results are similar to those on the synthetic data. In order to conduct the experiments with the variation of the number of tuples, we randomly sample a subset of tuples. The average audit time, the ratio of restricted queries and storage remain nearly unchanged when we vary the total number of tuples.

6 Conclusion

Most previous works focus on privacy issues over data from a single source. This paper formulates a problem called competitive privacy which considers privacy issues when sequence data is integrated from more than one source. Our proposed algorithm CCF rejects queries efficiently and guarantees no competitive privacy breach. In all experiments, the auditing step can be achieved within 0.04s and the ratio of the total number of restricted queries over the total number of queries is also small (within 0.15).

Acknowledgements: The research of Raymond Chi-Wing Wong is supported by HKRGC GRF 621309 and Direct Allocation Grant DAG08/09.EG01. The research of Eric Lo is supported by Hong Kong Research Grants Council GRF grant PolyU 525009E and PolyU internal grant 1-ZV5R.

References

1. D. P. Dobkin et al. Secure databases: Protection against user influence. *ACM Trans. Database Syst.*, 4(1):97–106, 1979.
2. E. Lo et al. OLAP on sequence data. In *SIGMOD Conference*, 2008.
3. G. Miklau et al. A formal analysis of information disclosure in data exchange. *J. Comput. Syst. Sci.*, 73(3):507–534, 2007.
4. P. Seshadri et al. Sequence query processing. In *SIGMOD*, 1994.
5. R. Motwani et al. Auditing SQL Queries. In *ICDE*, 2008.
6. R. Ramakrishnan et al. SRQL: Sorted Relational Query Language. In *SSDBM*, 1998.
7. R. C.-W. Wong and E. Lo. Analyzing integrated sequence data in a competitive environment. In *http://www.cse.ust.hk/∼raywong/paper/analyzingIntegratedSequence-tech.pdf*, 2009.