Supporting Pattern-Preserving Anonymization For Time-Series Data

Lidan Shou, Xuan Shang, Ke Chen †, Gang Chen, Chao Zhang
College of Computer Science, Zhejiang University, China

Abstract—Time-series is an important form of data available in numerous applications and often contains vast amount of personal privacy. The need to protect privacy in time-series data while effectively supporting complex queries on them poses non-trivial challenges to the database community. We study the anonymization of time-series while trying to support complex queries, such as range and pattern matching queries, on the published data. The conventional k-anonymity model cannot effectively address this problem as it may suffer severe pattern loss. We propose a novel anonymization model called (k,P)-anonymity for pattern-rich time-series. This model publishes both the attribute values and the patterns of time-series in separate data forms. We demonstrate that our model can prevent linkage attacks on the published data while effectively support a wide variety of queries on the anonymized data. We propose two algorithms to enforce (k,P)-anonymity on time-series data. Our anonymity model supports customized data publishing, which allows a certain part of the values but a different part of the pattern of the anonymized time-series to be published simultaneously. We present estimation techniques to support query processing on such customized data. The proposed methods are evaluated in a comprehensive experimental study. Our results verify the effectiveness and efficiency of our approach.

Index Terms—Privacy, Anonymity, Pattern, Time-series

1 Introduction

Time-series has long been considered one of the most important types of data available in both nature and human society. In recent years, the popularity of sensor networks, RFIDs, and wireless positioning equipments has further driven the production of time-series data to unprecedented volume and complexity. The publicity of these data on the Internet has nurtured the most creative applications ranging from financial analysis to social community tracking and partner matching. However, such massive data also implies vast amount of privacy, which, if not appropriately protected, may become exploited as a source for abuses and crimes.

Privacy protection in the publication of time-series is a challenging topic mostly due to the complex nature of the data and the way that they are used. In particular, the spectrum of frequently-used "complex" queries on time-series covers not only range queries on the attribute values at specified time instants but also pattern similarity queries which treat each sequence more globally. Unfortunately, it is no trivial task to support such variety of queries without disclosing the sensitive information of individuals.

Specifically, we consider an essential problem of *anonymizing* time-series while trying to support the queries mentioned above. For example, in a de-identified database of monthly sales of companies, users may issue

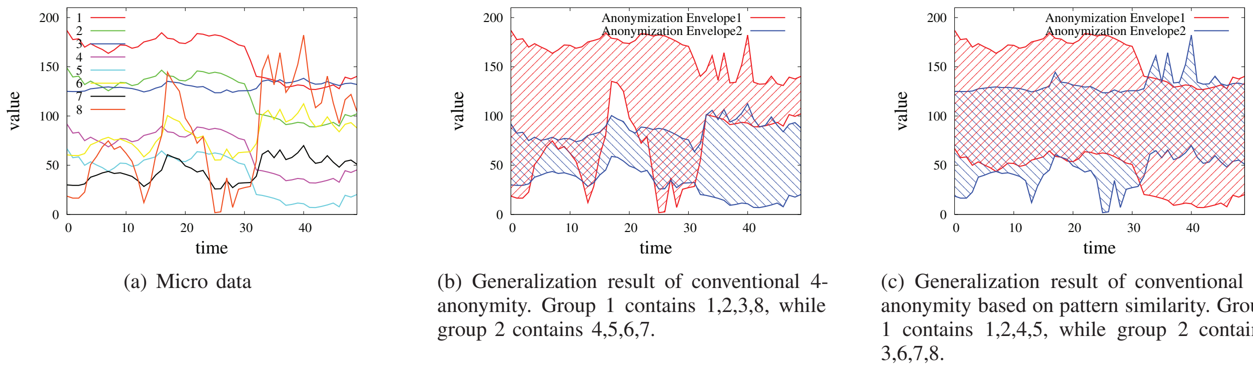
1) Range queries which specify *value* conditions, such as: $select * from\ dataset\ where\ sales.December \in [1\ million, 1.2\ million]$, or
2) Pattern matching queries which rely on the definition of *pattern similarity*, such as: *Given time series* q, *select* r *from dataset where* $similarity(r,q) > threshold$ (or $distance(r,q) < \delta$).

Meanwhile, it is critical to ensure that no identifiers of these companies will be disclosed. However, the time-sensitive attribute values and their patterns can be used as strong quasi-identifiers (QI) to launch linkage attacks which re-identify some of the records (time-series). For instance, an adversary may learn from the external sources that the monthly sale of the victim happens to be between 1 and 1.2 million in December, among the very few who can achieve that figure. Then (s)he could issue the above range query on the published table and the link between the victim's ID and the sensitive attribute values is easily established. Similarly, the pattern matching query which retrieves few results from the database could also be used for attacking.

The above example reveals the difficulties encountered when anonymizing time-series: On one hand, the instant values and global patterns of time-series have to be retained in the published data as much as possible to support various queries. On the other hand, the linkage attacks based on knowledge of values, patterns, or both, have to be prevented.

The conventional solution to prevent linkage attacks is to enforce k-anonymity [21], [14] on the published database, so that each record has its QI attributes identical to at least $k-1$ other records. Although conventional k-anonymity can be used to resist linkage attacks, it cannot effectively preserve the patterns, which are critical for performing queries on time-series. A few previous studies [17], [20] have proposed methods to anonymize sequences or trajectories by k-anonymity. Nergiz et al. in [17] proposed a method

† *Corresponding Author: chenk@zju.edu.cn*
This work was supported in part by the National Science Foundation of China (Grant No. 60970124, 60903038, and 61170034). A preliminary version of this work appeared as a short paper in CIKM'2010.



(a) Micro data

(b) Generalization result of conventional 4-anonymity. Group 1 contains 1,2,3,8, while group 2 contains 4,5,6,7.

(c) Generalization result of conventional 4-anonymity based on pattern similarity. Group 1 contains 1,2,4,5, while group 2 contains 3,6,7,8.

Fig. 1. Existing k-anonymity methods

called perturbation-driven k-anonymity. As this method publishes data by firstly enforcing k-anonymity and then reconstructing randomly a trajectory from the anonymized form, it is prone to significant pattern loss. Another work in [20] proposed the so-called pattern-preserving k-anonymity for sequences of a finite item set, and employed a prefix tree to realize anonymization. However, the pattern similarity considered in this work is only limited to exact string match. It is therefore not applicable for more general time-series data. In summary, we believe that no previous work has adequately addressed the anonymization of time-series to answer the most frequently-used (range and similarity) queries in the published database.

We exemplify the problems of conventional k-anonymity on time-series data in Figure 1(a)-1(c). Figure 1(a) shows a sample dataset containing 8 original time-series, identified by $\{1, 2, ..., 8\}$, where the QI attributes are the values at time $0, 1, \ldots, 50$. To enforce k-anonymity, each QI attribute value is generalized to a range such that each record is contained within a so-called *anonymization envelope* (AE), which also contains at least k-1 other records. The formation of an AE may be based on metrics such as Euclidean distance or pattern similarity. Figure 1(b) and Figure 1(c) illustrate the AEs (groups) of the same database using the Euclidean and pattern similarity distances respectively, given a $k = 4$. AnonymizationEnvelope1 denotes the AE of group 1. AnonymizationEnvelope2 are defined similarly. Unfortunately, the patterns in the original data might be significantly weakened, if not totally lost, in the generalized output, making the result useless for similarity queries. As shown in Figure 1(b), the patterns implied in each group are rather unclear. While in groups formed by pattern similarity (Figure 1(c)), the patterns shown in the upper and lower boundaries of each envelope may happen to be similar (e.g. group 1), we can however not necessarily derive any implied patterns for those contained within these two boundaries, as group 2 indicates. Another issue with anonymization based on pattern similarity is that the ranges on attribute values of an envelope might be very large, thus significantly harming the query accuracy.

We advocate an approach which explicitly preserves and publishes the patterns of time-series in a separate form of data, namely *pattern representations* (PRs). This approach will inevitably introduce a series of new problems, such as the definition, generalization, representation, and measurement of patterns. It is even more important to observe that pattern representations, too, may be exploited for linkage attacks. Thus we need to consider the possible attacks after publishing the PRs. As our approach handles the generalization of the time-series and their respective patterns separately, the probability of privacy breach after the anonymization and the information loss during this process must be re-examined. Intuitively, the objective of our approach is to constrain the breach probability under a specified value while trying to minimize the information loss.

By segregating the patterns, we propose a novel model called (k,P)-anonymity, which adopts a new privacy constraint P against linkage attacks based on pattern representations. This model ensures anonymity on two levels. On the first level, k-anonymity is required for time-series in the entire database. That means the records in the published database can be grouped by the quasi-identifier attribute values, and each group should contain at least k records. On the second level, P-anonymity is required for the pattern representations associated with each record in a same group. Specifically, each group can be divided into subgroups, each of which contains at least P records having identical PRs. As a result, the (k,P)-anonymity model is able to resist unified attacks based on both the attribute values and the patterns of the time-series, with a worst disclosure probability of $1/P$.

We propose two algorithms for generating results conforming to (k,P)-anonymity from any arbitrary time-series databases. The first one is a naive algorithm which extends the conventional k-anonymity algorithm by further partitioning each anonymity group. The naive algorithm produces coarse pattern representations due to limited partitioning space. The second algorithm, called KAPRA, uses a top-down group partitioning algorithm to split the entire database into a hierarchy of nodes as possible units conforming to the second level P-anonymity of PRs. These nodes are subsequently merged to generate the first level k-anonymity groups. The KAPRA algorithm is able to produce finer pattern representations conforming to (k,P)-anonymity. To resist the homogeneity attack and enhance the privacy preservation ability of our model, we also discuss a technique to extend (k,P)-anonymity by implementing the *l*-diversity on the sensitive attributes of time-series.

Our contributions are summarized as following:

- Relying on a very generic definition to patterns, we propose a novel anonymization model called (k,P)-anonymity for pattern-rich time-series. This model publishes both the *attribute values* and the *patterns* of time-series in separate data forms. We demonstrate that our model can prevent linkage attacks on the published data while effectively supporting a wide variety of queries on the anonymized data.
- We analyze the disclosure probability and the information loss metrics for our proposed (k,P)-anonymity.
- Two algorithms are designed to enforce (k,P)-anonymity on time-series data.
- We also propose the techniques for supporting customized data publishing, which allows the values and PRs to be published from different subsets of the QI attributes.
- The above algorithms are evaluated in a comprehensive experimental study.

The remainder of this paper is organized as follows. We briefly review the related work in Section 2. In Section 3, we introduce a generic pattern definition for time-series, identify the linkage attacks and the privacy requirements to the published data. Subsequently we describe the (k,P)-anonymity model and present measures of (k,P)-anonymity in Section 4. In Section 5, we present the algorithms and other implementation issues to enforce (k,P)-anonymity on an arbitrary dataset. As an extension to our method, Section 6 introduces the techniques for customized data publishing. The experimental results are reported in Section 7. Finally, Section 8 concludes the paper.

2 Related Work

In this section, we will summarize the existing works of partial information hiding, especially those related to time-series. The existing partial information hiding approaches can be divided into two categories, the *perturbation-based* approaches and the *partition-based* approaches.

Perturbation-based approaches protect data by adding noises. The noise should satisfy certain conditions or follow some kind of distribution to make the perturbed data have several common characteristics with original data, e.g, identical mean value. [19] [22] are based on perturbation. However, perturbation does not aim at linkage attack, which is the focus of our work.

Partition-based approaches first divide tuples of dataset into disjoint groups and then release some general information of each group. k-anonymity [14] and condensation [3] are two popular approaches in this category.

K-anonymity is an essential approach to privacy-preserving data publishing and generalization is the most popular approach of enforcing k-anonymity. But it has weaknesses when being applied on time-series data. After *QI* generalization, the pattern of time-series will be seriously distorted. Some previous work [20] [17] tried to apply k-anonymity on trajectories and sequences. Unfortunately, [17] does not achieve pattern preservation. The method in [20] is only limited to symbolic sequences, and cannot handle more general time-series. In addition, the prefix tree used in [20] is not applicable for real-valued sequences as it is possible to prune the whole prefix tree. We will thoroughly compare the performance of these two works with our model in Section 7.

The works in [2] and [16] are similar to [20], but [2] protects sensitive patterns by changing some elements of the sequence, which makes it possible to cause the most significant pattern feature lost. [16] assumes limited background knowledge which may compromise its privacy protection ability.

Some privacy principles, such as *l*-diversity [15] and *t*-closeness [12], are proposed based on k-anonymity to resist homogeneity attack. *l*-diversity prevents uniformity and background knowledge attacks by ensuring that at least *l* sensitive attribute values are well-represented in each k-group so that the probability to associate a tuple with a sensitive attribute value is bounded by $\frac{1}{l}$.

In [3], the authors proposed a condensation approach. The key difference between condensation and k-anonymity methods is that the former works with pseudo-data rather than the original records. The limitation of condensation is that it cannot preserve the correlations of attributes for individual data. That makes it unsuitable for time-series.

Micro-aggregation [18] can be used to prevent linkage attacks on time-series. It first clusters the micro data and then replaces original values with the centroid of the clusters. However, micro-aggregation will substantially change the dataset size, which may cause trouble for some quantity-sensitive applications. In addition, the data published by micro-aggregation also suffer pattern loss in an uncontrolled manner.

3 Preliminaries and Problem Definition

In this section, we first introduce preliminaries of conventional k-anonymity and the definition of patterns. Then we formulate the problem that we try to solve.

3.1 Conventional K-anonymity of Time-Series

The conventional k-anonymity of time-series assumes that each original time-series (record) r in a database T contains the following three parts of data:

- an *identifier id*;
- a set of *quasi-identifier (QI) attributes* at n different but typically consecutive time instants, denoted by $QI = \{A_1, A_2, \ldots, A_n\}$;
- a set of *sensitive attributes* which are denoted by A_S as an entirety.

The sensitive attributes are those whose values for any particular individual must be kept secret from people who have no direct access to the original data.

TABLE 1
Micro Data(unit:thousand)

TupleID	Name	2005	2006	2007	2008	2009	2010	2011(A_S)
1	Alice	170	175	188	197	213	221	200
2	Bob	145	157	165	177	204	196	180
3	Cathy	176	181	147	134	125	112	160
4	David	98	120	125	132	151	161	110
5	Jane	117	107	87	74	51	56	85
6	Lily	32	54	59	67	96	101	90
7	Mary	88	93	56	43	20	25	55
8	Steve	71	63	47	38	43	20	46

During the anonymization, all records are first de-identified, and then the QI attributes are transformed to prevent linkage attacks (reidentification of A_S). Specifically, each QI attribute A_i is generalized into a value range $R_i = [r_i^-, r_i^+]$ so that the value of A_i is in $[r_i^-, r_i^+]$. The conventional k-anonymity ensures that all the QI attribute values of each record in the published data, namely $R_1, \ldots, R_n$, are identical to at least $k-1$ other records. In a global picture, the anonymization creates in effect a number of *anonymization envelopes* on the QI attributes. Each envelope, lower-bounded by sequence $(r_1^-, r_2^-, \ldots, r_n^-)$ and upper-bounded by $(r_1^+, r_2^+, \ldots, r_n^+)$, contains at least k original records. Records belonging to the same envelope are called an *anonymity group*. To avoid confusion we use term *k-group* to refer to such anonymity group.

The sensitive attributes are critical information for subsequent time-series analysis and are typically retained in their original form. It is worthwhile to mention that, for conventional k-anonymity, the value of A_S has no effect on the anonymization process. But advanced models of k-anonymity (such as *l*-diversity and *t*-closeness) consider its diversity and distribution in each k-group when performing anonymization.

We now illustrate how the conventional k-anonymity fails to provide support for queries by *pattern* conditions. Table 1 and 2 show a micro data set of personal income and its anonymized version (The tuple ID of table 2 is deliberately kept for clarity). We regard the income of the past six years as the QI attributes, and that of the current year as the sensitive attribute. For a user interested in the correlation of the income of consecutive years, she may want to find out whether the income of 2006 is higher than that of 2005 for all records. She will sadly find the published data useless because the value ranges of year 2005 and 2006 significantly overlap each other on each record. In this particular case, the correlation among different QI attributes are not preserved.

Motivated by this example, we attempt to preserve and publish time-series patterns in a separate form of data in our approach. Apparently, this objective depends on the proper definition and representation of patterns.

3.2 Pattern Representation and Pattern Matching Queries

According to [6], patterns of time-series include trends and seasonalities. In whatever forms, a pattern can be taken as a set of features. Given a number of attributes of time-series, denoted by $A_1, \ldots, A_n$, a *feature* is defined as a correlation function f on these attributes.

$$f : (A_1, \ldots, A_n) \rightarrow Y$$

where Y is a domain of any arbitrary values. For example, one feature might be a function checking if the second attribute value is greater than the first one. In such case the feature is $f(A_1, A_2)$, and its possible values are "$>$", "$<$", or "$=$".

Definition *1: (Pattern)* A *pattern* of a time-series r is a feature vector of m correlation functions

$$\mathbf{p}(r) =< f_1, f_2, \ldots, f_m >,$$

where m is a system parameter. The patterns of two time-series are said to be *similar/equal*, if their respective feature vectors are similar/equal.

The above definition to pattern is very generic. In one special case, a pattern can be the time-series itself if we define $f_i = A_i$ for $i = 1, \ldots, n$. Therefore, we believe that such definition can virtually capture almost all kinds of correlations among the attributes of time-series. We note that our definition of time-series patterns is consistent with the one given in a more generic background [23].

Relying on the notion of pattern, we may define a pattern matching query as a *range* or *similarity* query in the feature space of patterns. A *pattern matching range query* (PRQ) retrieves time-series whose feature vectors satisfy the specified range predicates:

`SELECT r FROM T WHERE` $f_1(r) \in [v_1^-, v_1^+]$ *and* $f_2(r) \in [v_2^-, v_2^+]$ *and* $\ldots$ *and* $f_w(r) \in [v_w^-, v_w^+]$

where $[v_w^-, v_w^+]$ is the value range specified for each correlation function f_w. For example, given two correlation functions $f_1 = A_2 - A_1$ and $f_2 = \frac{A_2}{A_1}$, the following query retrieves records whose attribute A_2 is at least 1.5 times more than A_1 and the difference between the two is less than 50: `SELECT r FROM T WHERE` $f_1(r) \in (-\infty, 50)$ `and` $f_2(r) \in [1.5, +\infty)$. Note that a PRQ in the pattern feature space is different from a range query on the attribute values.

Likewise, a *pattern matching similarity query* (PSQ) can be defined with a distance metric D in the feature space. Given a tolerance $\delta \geq 0$ and a querying sequence q, a PSQ retrieves the time-series which satisfy $D(r, q) \leq \delta$.

`SELECT r FROM T WHERE` $D(r, q) \leq \delta$

In practice, however, the number of correlation functions in a pattern may be very large. For example, a "difference" pattern which compares the values between any two of the n attributes of a sequence may contain $m = C_n^2$ components in its feature vector. It is costly to publish such large feature vectors which apparently contain much information redundancy. Thus, more concise data formats are needed for publishing patterns.

We introduce the notion of *pattern representations* for pattern publishing.

Definition *2: (Pattern Representation)* Given the m correlation functions defined for patterns, a *pattern representation (PR)* of a time-series is an entity which

TABLE 2
Traditional 4-anonymity(unit:thousand)

TupleID	GroupID	2005	2006	2007	2008	2009	2010	2011(A_S)
1	1	[98-176]	[120-181]	[125-188]	[132-197]	[125-213]	[112-221]	200
2	1	[98-176]	[120-181]	[125-188]	[132-197]	[125-213]	[112-221]	180
3	1	[98-176]	[120-181]	[125-188]	[132-197]	[125-213]	[112-221]	160
4	1	[98-176]	[120-181]	[125-188]	[132-197]	[125-213]	[112-221]	110
5	2	[32-117]	[54-107]	[47-87]	[38-74]	[20-96]	[20-101]	85
6	2	[32-117]	[54-107]	[47-87]	[38-74]	[20-96]	[20-101]	90
7	2	[32-117]	[54-107]	[47-87]	[38-74]	[20-96]	[20-101]	55
8	2	[32-117]	[54-107]	[47-87]	[38-74]	[20-96]	[20-101]	46

1) can be obtained by a transformation $\mathcal{M}(\cdot)$ from the time-series itself; and
2) can be transformed to a pattern determinedly.

Given a time-series r, we denote by $PR[r] = \mathcal{M}(r)$ its pattern representation.

The second item in the above ensures that a pattern can be *reconstructed* from its PR. Therefore, the pattern matching range/similarity queries defined previously can be performed on the reconstructed patterns.

When generating PRs, transformation $\mathcal{M}(\cdot)$ would certainly incur information loss, which is called the *pattern loss*. As a result, the reconstructed pattern might be distorted, leading to inaccuracy in the subsequent pattern matching queries. Therefore, a key requirement for the PR transformation $\mathcal{M}(\cdot)$ is to minimize the pattern loss, which is measured by the distortion after pattern reconstruction.

3.3 Data Publishing and Privacy Attacks

Before introducing our anonymity model, we shall look at both the data that we try to publish and the possible attacks on such data. We shall first assume a simple and yet typical scenario called *Full Data Publishing (FDP)*, where the values and PRs to be published are generated from *all* attributes in *QI*. Our main algorithms proposed in the sequel focus on FDP. However, we will also address *Customized Data Publishing (CDP)* as an extension in Section 6, where the values and PRs are published from different subsets of the QI attributes.

Definition *3: (Full Data Publishing)* Given $QI = \{A_1, \ldots, A_n\}$, each record r^* in the published database T^* contains three parts:

- the *QI value ranges* $(R_1, \ldots, R_n)$, which define an anonymization envelope across the n QI attributes,
- the *QI pattern representation* $PR[r]$, obtained from the n QI attributes of r,
- the *sensitive information* A_S, which is same as its counterpart in T.

Subsequently, the *background knowledge* of an adversary may come from (1) the values of the QI attributes, denoted by K_v, and (2) the whole QI pattern or part of it, denoted by K_p. Thus, we are able to derive the following three linkage attacks from the two types of background knowledge:

- Attacks based on K_v;
- Attacks based on K_p;
- Attacks based on $K_v \bigcup K_p$, also called the unified attack.

The quantitative measure of privacy attacks relies on the notion of *privacy breach probability*. Regarding the above three linkage attacks, the privacy breach probability is defined as follows.

Definition *4: (Privacy breach probability)* For one time-series Q being considered as a target in database T, the privacy breach probability for Q, denoted by $P_{breach}[Q]$, is the probability that an adversary can infer from T^* that $Q.id$ becomes associated to the value of $Q.A_S$ based on the background knowledge $K_v \bigcup K_p$. If $P_{breach}[Q] > thr$, where thr is a system parameter, we say that the privacy of Q is breached.

Therefore, the problem that we tackle is to generate a T^* which

- Retains as much information from the micro data as possible, in both the QI value ranges and the QI pattern representation;
- Ensures $P_{breach}[r] \le thr$ for all records $r \in T$.

4 The (k,P)-anonymity model

In this section, we introduce the (k,P)-anonymity model to address the problem described in Section 3.3. We also look at the characteristics of the proposed model. In the end of this section, we propose a general framework to publish data conforming to (k,P)-anonymity.

4.1 The Anonymity Model

We now present the (k,P)-anonymity model. Our approach assumes that each time-series is published in three components, namely the *QI value ranges*, the *QI pattern representation*, and the *sensitive information*. For clarity of presentation, the (k,P)-anonymity model can be described as a conceptual extension of the conventional k-anonymity. Nevertheless, the algorithm to enforce (k,P)-anonymity does not have to rely on the conventional k-anonymity algorithm.

As Figure 2 illustrates, our model ensures anonymity on two levels. On the first level, the QI attributes are generalized to fulfill the conventional k-anonymity, regardless of the QI pattern representation. The results of the generalization contain a number of partitions known as the *k-groups*. We note that the QI value ranges are analogous to those in conventional k-anonymity. The second-level anonymity considers records in each k-group. For any record r in a k-group, if there exist at least $P - 1$ other records which

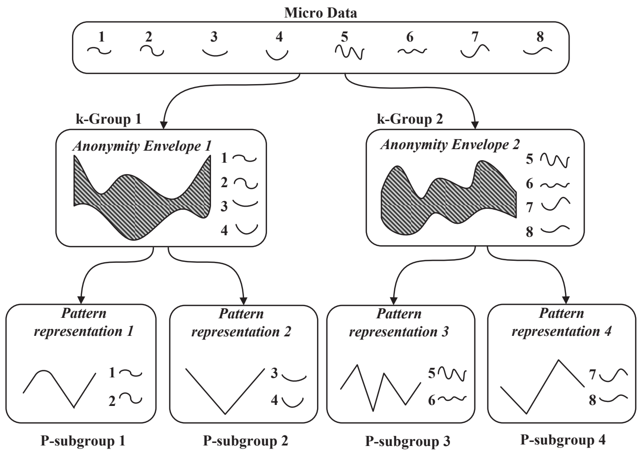


Fig. 2. The k-groups and P-subgroups of (k,P)-anonymity

have the same pattern representation as r, we say that *P-anonymity* is enforced for this k-group. As a result, we can partition the k-group further into subgroups, each of which contains at least P records having the identical PR.

A generalized database T^* is said to satisfy *(k,P)-anonymity* if P-anonymity is enforced on all k-groups. (k,P)-anonymity can be defined as follows:

Definition 5: *((k,P)-anonymity)* Let T be a database of time-series, and $A_1, \ldots, A_n$ being the QI attributes. A published database T^* is said to satisfy (k,P)-anonymity, if T^* meets the following two requirements:

- k-requirement: each anonymization envelope $AE = (R_1, \ldots, R_n)$ appears for at least k times in the entire database;
- P-requirement: consider any k-group G of time-series having the identical anonymization envelope, if any time-series $r \in G$, there are at least $P-1$ other time-series in G having the same QI pattern representation as $PR[r]$.

Table 3 illustrates one possible output scheme conforming to (k,P)-anonymity for the running example. In the published dataset, to preserve the pattern information for each time-series, there is an additional PR column which adopts an alphabetic string representation. QI attributes are generalized based on the formed k-group ($k = 4$ in our example). In each k-group, the PR column conforms to P-anonymity ($P = 2$ in our example). We shall discuss the form of PR later in Section 5.1. One thing to note here is that the grouping scheme in Table 3 is different from the one used for conventional k-anonymity, as shown in Table 2. The reason will be explained in Section 5.

We can easily extend (k,P)-anonymity to adopt *l*-diversity by adding data transformation on the sensitive attributes, thus achieving stronger privacy protection. The details about this extension is given in Section 5.4.

4.2 The Utility Measures

In this subsection we will introduce the utility measures of (k,P)-anonymity model, including the breach probability, which stands for the privacy protection ability, and the information loss, which represents the utility of published data. There are two types of information loss, *instant value loss* (VL), and the *pattern loss* (PL). We will separately present them in Section 4.2.2 and Section 4.2.3.

4.2.1 The Breach Probabilities

The evaluation of breach probabilities is based on the assumption that the sensitive attribute information A_S is unique across each k-group. We only discuss the probability for the most general *unified linkage attack*, which is based on $K_v \bigcup K_p$. Those of the other two attacks can be derived similarly.

For any target time-series Q, we can obtain a possible set of Q, denoted as $PS(Q)$, from T^* which contains the time series whose anonymization envelopes encompass the original values of Q and have pattern representations identical to $PR[Q]$. Let $PS_{Q.A_S}(Q)$ be the subset of $PS(Q)$ which contains time-series whose sensitive attribute values equal to $Q.A_S$. From Definition 4 we know that the privacy breach probability could be regarded as the proportion of $PS_{Q.A_S}(Q)$ in $PS(Q)$, which means $P_{breach}[Q] = \frac{|PS_{Q.A_S}(Q)|}{|PS(Q)|}$. We define $b = |PS_{Q.A_S}(Q)|$, and denote by $e[K_v]$ ($e[K_v] \geq 1$) the number of k-groups whose anonymization envelopes encompass K_v. $e[K_p]$ and $e[K_v \bigcup K_p]$ are defined similarly.

In unified linkage attack, the adversary has both K_v and K_p about Q, then $|PS(Q)| = P * e[K_v \bigcup K_p]$ since there should be at least P indistinguishable time-series based on $K_v \bigcup K_p$ in each k-group that possibly contains Q. Therefore we have

$$P_{breach}^{K_v \bigcup K_p}[Q] = \frac{b}{P \cdot e[K_v \bigcup K_p]}. \tag{1}$$

The calculation of $P_{breach}^{K_v}[Q]$ (attacks based on K_v) and $P_{breach}^{K_p}[Q]$ (attacks based on K_p) is similar to Equation 1. Under our assumption, in the worst case, $P_{breach}^{K_v}[Q] = P_{breach}^{K_p}[Q] = P_{breach}^{K_v \bigcup K_p}[Q] = 1/P$ ($P > 1$). Based on the privacy breach probability threshold *thr* defined in Section 3, we require $1/P < thr$. Therefore, if $P > 1/thr$, our anonymity model can resist the three linkage attacks proposed in Section 3.

4.2.2 Instant value loss metric

In order to make the published data as useful as possible, it is required to reduce the instant value loss as much as possible. We adopt a loss measure based on the anonymization envelope of each group. For a time-series Q belonging to QI group G, the anonymization envelope of G has a lower bound $(r_1^-, r_2^-, \ldots, r_n^-)$ and an upper bound $(r_1^+, r_2^+, \ldots, r_n^+)$. The instant value loss of Q is therefore given by

$$VL(Q) = \sqrt{\sum_{i=1}^{n} (r_i^+ - r_i^-)^2 / n} \tag{2}$$

For database T, $VL(T)$ is obtained by summing up the instant value losses of all its members.

TABLE 3
A published dataset T^* conforming to (k,P)-anonymity(unit:thousand)

TupleID	GroupID	2005	2006	2007	2008	2009	2010	PR	2011(A_S)
1	1	[117 – 176]	[107 – 181]	[87 – 188]	[74 – 197]	[51 – 213]	[56 – 221]	*aabbcc*	200
2	1	[117 – 176]	[107 – 181]	[87 – 188]	[74 – 197]	[51 – 213]	[56 – 221]	*aabbcc*	180
3	1	[117 – 176]	[107 – 181]	[87 – 188]	[74 – 197]	[51 – 213]	[56 – 221]	*ccbbaa*	160
4	2	[32 – 98]	[54 – 120]	[47 – 125]	[38 – 132]	[20 – 151]	[20 – 161]	*abbbcc*	110
5	1	[117 – 176]	[107 – 181]	[87 – 188]	[74 – 197]	[51 – 213]	[56 – 221]	*ccbbaa*	85
6	2	[32 – 98]	[54 – 120]	[47 – 125]	[38 – 132]	[20 – 151]	[20 – 161]	*abbbcc*	90
7	2	[32 – 98]	[54 – 120]	[47 – 125]	[38 – 132]	[20 – 151]	[20 – 161]	*bbbaaa*	55
8	2	[32 – 98]	[54 – 120]	[47 – 125]	[38 – 132]	[20 – 151]	[20 – 161]	*bbbaaa*	46

4.2.3 Pattern loss metric

The pattern loss metric should be defined based on the feature vector $\mathbf{p}(\cdot)$ as we proposed in Section 3. For any time-series Q, we can obtain its original feature vector $\mathbf{p}(Q)$, which represents the pattern information embodied in the original time-series. Meanwhile, we can obtain from $PR[Q]$ the reconstructed feature vector $\mathbf{p}^*(Q)$ which represents the pattern information preserved in $PR[Q]$. Therefore, the pattern loss can be measured by the distance between $\mathbf{p}(Q)$ and $\mathbf{p}^*(Q)$, namely

$$PL(Q) = distance(\mathbf{p}(Q), \mathbf{p}^*(Q)) \tag{3}$$

where $distance(\cdot)$ is a distance measure defined in the feature vector space of patterns. For simplicity we use the well-known cosine distance to calculate the pattern loss. However, it can easily be replaced by other distance measures. For a whole database T, $PL(T)$ is obtained by summing up the pattern loss of all its members.

4.3 Approaches for Enforcing (k,P)-Anonymity

Now we will look at the method to enforce (k,P)-anonymity on an arbitrary micro dataset. Our target is to minimize the information loss while respecting the constraints on the breach probabilities. It can be proven that a global optimal solution requires combinatorial computation cost (Proof omitted to save space). Therefore, we will consider more efficient near-optimal solutions in the sequel.

Motivated by the conventional k-anonymity, one possible solution for enforcing (k,P)-anonymity is to employ a *top-down* clustering-like framework as described in the following:

1) Generate first-level *k-groups* from the micro dataset.
2) For each k-group, extract PRs from micro data based on the chosen PR form. The extracted PRs should minimize the pattern loss while respecting the P-requirement within its own k-group;
3) For each k-group, generate *P-subgroups* based on the PRs;

Step 2 is a challenging task and highly dependent on the PR form being used. Different PR forms may lead to very different implementations of this step. In whatever forms, the granularity of PR should be carefully tuned to achieve the optimization target of this step. The top-down approach is easy to understand as it can be regarded as an extension to the existing k-anonymity approach.

Alternatively, we can employ a *bottom-up* framework to form P-subgroups from individual records first, and then build k-groups. The bottom-up approach is described in the following:

1) Extract PRs from the micro data. The extracted PRs should minimize the pattern loss while respecting the P-requirement in the entire dataset;
2) Form the second-level *P-subgroups* based on PRs;
3) Form the first-level *k-groups* based on the P-subgroups formed in Step 2.

In this approach, Step 1 is similar to the second step of the *top-down approach*. The process of Step 2 is highly dependent on the output of Step 1.

In the next section, we shall propose for each framework one algorithm – The Naive algorithm as a top-down approach; while the KAPRA algorithm as a bottom-up one.

5 Algorithms and Implementation Issues

This section presents our algorithms which transform an original dataset and produce output conforming to (k,P)-anonymity. We start by introducing a specific PR extraction technique based on the well-known SAX [13] representation of time-series. Then we propose a naive *top-down* algorithm as an extension to a conventional k-anonymity algorithm. Subsequently we propose a more advanced algorithm called KAPRA (literally for K And P-Reinforced Anonymity) following the *bottom-up* framework discussed in Section 4.3.

5.1 Computing Pattern Representations

When computing pattern representations, our main purpose is to achieve minimal pattern loss in the published table. For the convenience of publishing, we need to find a pattern representation method which is easy to understand. Most importantly, due to the P-requirement and the needs of minimizing pattern loss, the accuracy of *PR* should be able to be tuned. Besides, as neither collusion nor data republication is considered in this paper, the *PR* is supposed to be general-purpose so that it could support all different usages of the published data.

Numerous techniques have been proposed in the literature for representing time-series for different needs [13] [10] [9] [8]. The requirements and considerations discussed in the above naturally lead us to choose SAX [13]. Apart from these reasons, the discretization nature of SAX allows one to better exploit the data structure making use of the enormous tools and methods developed on symbolic space [13].

SAX uses a sequence of alphabets, for example "baabccbc", to represent a time-series. Given a set of alphabets $\Sigma_{level} = \{\alpha_1, \ldots, \alpha_{level}\}$, a time-series can be converted to an alphabet sequence as described in the following:

1) First, we need to specify an integer parameter *level* which controls in effect the granularity of the resultant SAX representation (the string).
2) Second, we normalize the time-series to have a mean of 0 and standard deviation of 1.
3) Third, each value in the normalized series is compared against a list of predefined "break points" $(\beta_1, \ldots, \beta_{level-1})$, which are actually the $level - 1$ number of quantiles of the Gaussian distribution. A value falling in range $[\beta_{j-1}, \beta_j)$ is given an alphabet of α_j.

We use $PR_{sax}[r] = sax_1 \ldots sax_n$ to denote the n alphabets in the SAX representation of time-series r.

The advantage (and also the limitation) of SAX is its independence on the pattern correlation functions. On one hand, we can obtain the PRs without knowing the correlation functions beforehand. This is desirable because it can easily capture many pattern features even when the pattern functions are unknown. On the other hand, if the SAX representations obtained are known to be poor in reproducing the pattern features, little can be done to improve the reconstructed patterns. However, we believe that SAX is general-purpose and therefore good enough to fulfill the vast needs of pattern matching queries.

Considering our need to enforce (k,P)-anonymity, it is a tricky problem to set an appropriate *level* for each time-series. To fulfill the P-requirement, for each time-series r we have to find other $P - 1$ time-series whose *PR*s (and *level*s) are identical with $PR[r]$. However, for preserving the patterns, we shall try to maximize the *level* of each time-series. These two requirements are contradictory because increasing the *level*s will apparently make the P-requirement harder to satisfy. In fact, the procedure of tuning the *level*s is tightly coupled with the anonymization algorithms and will be described in the subsequent texts.

A pattern vector $\mathbf{p}^*(r)$ can be reconstructed from its SAX representation as described following. First we will reconstruct from $PR[r]$ a numerical sequence r', which approximates the normalized sequence of r. For each element sax_i in $PR_{sax}[r]$, if $sax_i = \alpha_j$ where α_j is an alphabet in Σ_{level}, the respective r'_i is given by the Gaussian function at the probabilistic median of β_{j-1} and β_j (i.e. $Gaussian(\frac{j-1}{level} + \frac{1}{2*level})$). Then $\mathbf{p}^*(r)$ can be calculated as $\mathbf{p}(r')$. Since parameter *level* is needed for reconstruction, we require the *level* of each $PR_{sax}[r]$ to be kept in the PR column of T^*.

5.2 The Naive Algorithm

The naive solution to enforce (k,P)-anonymity includes two phases: (1) Firstly, we employ a *top-down clustering* procedure similar to [25] to ensure k-anonymity of the dataset. We note that this algorithm can be replaced by other existing k-anonymity algorithms at trivial cost. (2) Subsequently, an additional *create-tree* procedure is performed for each of the k-groups formed in the first phase. For each k-group G, we need to split it into nodes representing P-subgroups and choose an appropriate level of PR for each node satisfying the P-requirement. As for each time-series the pattern loss decreases when the SAX level increases, we try to maximize the SAX level as long as the P-requirement is satisfied. This procedure can be performed recursively in a top-down manner to generate a tree-structure.

The details of the *create-tree* phase are described as follows:

(1) *Initialization* Initially the whole k-group G is regarded as a root node for splitting, with the SAX level set to 1 ($level = 1$ represents the coarsest granularity for pattern representation). We denote the highest-granularity level of SAX that we allow as *max-level*.

Each tree node N has five attributes:

- *level*: the current SAX level of N;
- *PR*: the SAX-PR of this node, depending on the current *level*;
- *members*: the time-series contained in N, all having SAX-PR as *N.PR*;
- *size*: the number of time-series contained in N.
- *label*: each node has three possible labels: *bad-leaf*, *good-leaf* or *intermediate*. A label of *intermediate* indicates that N is not a leaf node. A *good-leaf* indicates that N is a leaf node whose $size \geq P$, while a *bad-leaf* indicates one with its $size < P$.

(2) *Node splitting* The node splitting process is guided by attempts to refine the PRs of the records in a node. For each tree node N, all its members should have identical PR *N.PR* under *N.level*. However, an increment in the level leads to changes in PRs. Therefore, the members of N may have different PRs at $N.level + 1$. Thus by increasing the level of N, we can split N into a few partitions of records, so that each partition captures records of the same PR at $N.level + 1$.

Starting from the root, we handle node N in a recursive procedure:

- If $N.size < P$, then the node is labeled as bad-leaf and the recursion terminates.
- Otherwise if $N.level = max\text{-}level$, then the node is labeled as good-leaf and the recursion terminates.
- Else if $P \leq N.size < 2*P$, we try to maximize the level of N as long as all records of N have the identical PR. The node is labeled as good-leaf and the recursion terminates. We try to avoid node splitting in such case for two reasons: First, this node is a good candidate for the resulting P-subgroups; second, a split on this node will for sure generate at least one bad-leaf, which will subsequently increase the burden of the next step.
- Otherwise, we need to check if node N has to be split. The checking relies on a tentative split performed on N. Suppose that, by increasing the level of N, N is

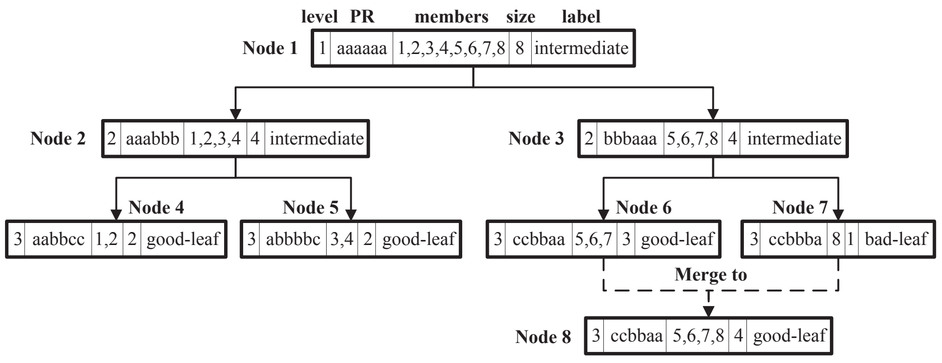

level PR members size label
Node 1 | 1 | aaaaaa | 1,2,3,4,5,6,7,8 | 8 | intermediate
Node 2 | 2 | aaabbb | 1,2,3,4 | 4 | intermediate
Node 3 | 2 | bbbaaa | 5,6,7,8 | 4 | intermediate
Node 4 | 3 | aabbcc | 1,2 | 2 | good-leaf
Node 5 | 3 | abbbbc | 3,4 | 2 | good-leaf
Node 6 | 3 | ccbbaa | 5,6,7 | 3 | good-leaf
Node 7 | 3 | ccbbba | 8 | 1 | bad-leaf
Merge to
Node 8 | 3 | ccbbaa | 5,6,7,8 | 4 | good-leaf

Fig. 3. The splitting tree structure of a k-group which contains 8 time-series denoted by {1,2,..,7,8}, where $P = 2$. Node6 and Node7 will merge into Node8 in Step 3. The final P-subgroups are denoted by Node4, Node5 and Node8.

tentatively split into a number of child nodes. If all these child nodes contain fewer than P time-series, no real split is performed and the original node N is labeled as good-leaf and the recursion terminates on N. Otherwise, there must exist tentative child node(s) whose *size* $\geq P$, also called *TG-node(s)* (Tentative Good Nodes). The rest children whose *size* $< P$ are called *TB-nodes* (Tentative Bad Nodes), if any. If the total number of records in all TB-nodes under N is no less than P, we merge them into a single tentative node, denoted by $child_{merge}$, at the level of $N.level$. If the above tentative process produces nc tentative child nodes (including TB and TG) and $nc \geq 2$, N will really be split into nc children and then the *node splitting* procedure will be recursively invoked on each of them.

In summary, node splitting will happen only when the target node N satisfies the following two conditions:

- $N.size$ is no less than $2 * P$;
- A tentative split on N produces more than one child node with at least one having size $\geq P$.

The pseudo code of the node splitting procedure is given in Algorithm 1.

Algorithm 1: **Node splitting**

```
Data: tree node $N$, $P$, $max\text{-}level$
begin
    if $N.size<P$ then
        $N.label = bad\text{-}leaf$;
    if $N.level==max\text{-}level$ then
        $N.label = good\text{-}leaf$;
    if $P \leq N.size < 2 * P$ then
        $N.label = good\text{-}leaf$;
        Maximize $N.level$ without node split;
    else
        if $N$ can be split then
            if total size of all $TB$-nodes $\geq P$ then
                generate $child_{merge}$;
                $child_{merge}.level = N.level$;
                level of all TG-nodes is $N.level + 1$;
            else
                level of all child nodes is $N.level + 1$;
        else
            $N.label = good\text{-}leaf$;
end
```

(3) *Post-processing* The bad-leaf nodes labeled in the above recursive node splitting process would undergo an additional post-processing. First, all bad-leaf nodes are deleted from the tree and inserted into a heap structure sorted in ascending order of node size. Starting from the first node of the heap, each bad-leaf BL is then removed from the heap and its records are re-inserted (merged) into a good leaf L_N in the tree, which contains a PR having largest *pattern similarity* to that of BL. Ties are broken by choosing the one with smaller size. The new leaf node L'_N will retain the PR of L_N. The post-processing will proceed until no bad-leaf nodes exist. As the tree is generated from a k-group, the number of leaf nodes is typically small. Thus the cost of post-processing is trivial.

The above create-tree phase is exemplified in Figure 3, which takes all records in Table 1 as an initial k-group of 8 time-series to be split into P-subgroups with $P = 2$. When $level = 1$, the whole k-group is a root node (Node1) with $PR = aaaaaa$. As $2 * P < 8$, we need to increase *level* by 1, causing *PR* to change. Thus Node1 will be split to two nodes, namely Node2 and Node3. Each of the two contains 4 time-series. Once again, as $2 * P \leq 4$, *level* will be incremented and Node2 will be further split to Node4 and Node5. So will Node3 be split to Node6 and Node7. Now all the *size*s of the 4 new nodes are less than $2 * P$. Thus the recursive node splitting process stops and all four nodes at $level = 3$ will be marked as leaf nodes. Moreover, Node7 is a bad-leaf because its $size < P$. Therefore, we have to merge Node7 with its nearest leaf node Node6 in post-processing. This produces a new leaf node Node8. The final resulting P-subgroups are, as represented by their node IDs, Node4, Node5, and Node8.

The Naive algorithm has a computational complexity of $O(max\text{-}level * |T| + |T|^2)$. Although the algorithm is easy to implement, the search space of the node splitting procedure is rather limited. Thus, the pattern representations generated from the Naive algorithm are expectedly very coarse. To improve the quality of the generated PRs, we propose a

Algorithm 2: **Recycle bad-leaves**

```
Data: $P$, leaf-list, current-level, max-bad-level
Result: $P$-subgroup list
begin
    current-level = max-bad-level;
    while sum of all bad leaves' size $\geq P$ do
        if any bad leaves can merge then
            Merge them to a new node leaf-merge;
            if leaf-merge.size$\geq P$ then
                leaf-merge.label= good-leaf;
            else
                leaf-merge.label= bad-leaf;
        current-level − −;
    Suppress all time-series contained in bad leaves;
end
```

Algorithm 3: **Group formation**

```
Data: $PGL$, $k$, $P$
Result: Group list $GL$
begin
    for each P-subgroup that size $\geq 2 * P$ do
        Split it by top-down clustering;
    if any P-subgroup that size $\geq k$ then
        Add it into $GL$ and remove it from $PGL$;
    while $|PGL| \geq k$ do
        Find $s_1$ and $G = s_1$;
        while $|G| < k$ do
            Find $s_{min}$ and add $s_{min}$ into $G$;
        Remove all $P$-subgroups in $G$ from $PGL$ and put $G$ in $GL$;
    for each remaining P-subgroup $s'$ do
        Find corresponding $G'$ and add $s'$ into $G'$;
end
```

more advanced algorithm called KAPRA which aims at producing finer PRs.

5.3 The KAPRA Algorithm

Relying on a tree structure capturing the PRs of all time-series in T, the KAPRA algorithm generally partitions the whole dataset into P-subgroups first, and then forms k-groups from the P-subgroups. More specifically, the algorithm can be divided into three (bottom-up) phases:

- Create tree phase;
- Recycle bad-leaves phase;
- Group formation phase.

5.3.1 Create Tree Phase

In this phase, we produce and organize the PRs in a tree for all time-series in T respecting the P-requirement. In this way we can ensure the patterns of the time-series to be maximally preserved. This phase of KAPRA follows the same principle of *create-tree* phase in the Naive solution except the following differences:

1) The root node for split is the whole dataset T;
2) The post-processing step in the Naive algorithm is removed. All leaf nodes are saved in a *leaf-list*, and the bad-leaves are separately disposed in the *recycle bad-leaves* phase.

The computation complexity of this phase is $O(\textit{max-level} * |T|)$.

5.3.2 Recycle Bad-Leaves Phase

All time-series contained in the bad-leaf nodes generated in the previous phase have to be suppressed from T^* as otherwise the P-requirement will be violated. To avoid too much suppression, we would recycle most of the bad-leaf nodes by merging them with each other.

To achieve the highest PR level for each bad-leaf, we start the recycling process from the highest PR level among all bad leaves, denoted by *max-bad-level*. Initializing parameter *current-level* to *max-bad-level*, we gradually decrease the value of *current-level* until the number of all time-series contained in bad-leaf nodes becomes less than P. If two bad-leaf nodes BL_1 and BL_2 have the same level and PR, they can be merged into a new node, denoted by n_m, which has the same level and PR. If n_m contains no fewer than P time-series, it is marked a good-leaf, otherwise a bad-leaf.

After that, all time-series still contained in the bad-leaf nodes have to be suppressed from T^*. Since the *current-level* could decrease to 1, on which all time-series will have the same PR, the number of suppressed time-series will always be less than P. Each remaining good-leaf node is actually a P-subgroup. Thus we get a list of P-subgroups at the end of this phase. If we denote the number of bad-leaves as NUM_{BL}, the computation complexity of this phase is $O(\textit{max-bad-level} \cdot NUM_{BL})$. The respective pseudo code is illustrated in Algorithm 2.

5.3.3 Group Formation Phase

This phase creates k-groups from the P-subgroup list generated from the previous phases, while minimizing the instant value loss of each k-group. We denote by PGL the input P-subgroup list, by GL the output k-group list, and by $|X|$ the number of records contained in X. The greedy k-group formation phase is described in the following steps:

(Preprocessing) The preprocessing step partitions all large P-subgroups into smaller ones. We find all large P-subgroups $s_i \in PGL$ where $|s_i| \geq 2P$. Each s_i will be partitioned into new subgroups no smaller than P using a top-down partitioning method similar to the *top-down greedy search* algorithm proposed in [25]. This partitioning process is targeted at minimizing the *total instant value loss* in the partitions. The resultant partitions, each regarded as a new P-subgroup, will be added into PGL to replace s_i.

(Step 1) All P-subgroups in PGL containing no fewer than k time-series are taken as k-groups and simply moved into GL. (Note that after preprocessing, there may still exist P-subgroup s_i satisfying $k \leq |s_i| < 2P$.)

(Step 2) In the remaining P-subgroups in PGL, find the P-subgroup s_1 with the minimum instant value loss, and then create a new group $G = s_1$.

(Step 3) Find another P-subgroup $s \in PGL - s_1$, which, if merged with G, produces the minimal value loss $VL(G \bigcup s)$.

Formally, we need to find

$$s_{min} = \arg\min_{s \in PGL} VL(G \cup s),$$

and merge s_{min} into G.

(Step 4) Repeat Step 3 until $|G| \geq k$. G is then added into GL and its respective subgroups in PGL are removed.

(Step 5) Step 2-4 are repeated until the total remaining time-series in PGL are fewer than k. Each remaining P-subgroup $s' \in PGL$ will choose to join a k-group $G' \in GL$, which again minimizes the total instant value loss.

The computation complexity of this phase is $O(|PGL|^2)$. Its pseudo code is illustrated in Algorithm 3.

5.4 Sensitive Attribute Transformation

In the main text, we simply assume that each sensitive attribute value is unique in its k-group. Now we will look at a perturbation technique to adapt our model to l-diversity based on data transformation.

Our definition of l-diversity is motivated by [24] and is defined as follows. For any record $r \in T$, an adversary knowing the knowledge of $K_v \bigcup K_p$ can in the worst case reduce the search domain to the possible set of r, $PS(r)$, which is defined in Section 4.2.1. We define the set of time-series in $PS(r)$ having an identical sensitive attribute value v_i as the *equivalence class* of v_i, denoted by $EC(v_i)$. Suppose $\mathcal{S}(r) = \{v_1, v_2, ...v_m\}$ denotes all distinct sensitive attributes appearing in $PS(r)$. Then we say that T satisfies the l-diversity if $\frac{|EC(v_i)|}{|PS(r)|} \leq \frac{1}{l}$ is true for all $r \in T$ and $v_i \in \mathcal{S}(r)$.

If the l-diversity is not satisfied in T, then the following measures are taken to perturb its sensitive attributes. Given a record $r \in T$, for each sensitive attribute value v_i with $\frac{|EC(v_i)|}{|PS(r)|} > \frac{1}{l}$, we randomly choose x_i number of records from $EC(v_i)$, and replace their sensitive attribute value v_i with slightly perturbed ones. Number x_i and each perturbed value of v_i are determined as described below:

(1) Number x_i is set to

$$x_i = |EC(v_i)| - \lfloor \frac{|PS(r)|}{l} \rfloor$$

so that a larger EC will have more of its records perturbed. It can be proven that $0 < x_i < |EC(v_i)|$. The proof is given in the following.

Proof: To prove $0 < x_i < |EC(v_i)|$, we only need to prove $0 < \lfloor \frac{|PS(r)|}{l} \rfloor < |EC(v_i)|$.

From the above we know that $|PS(r)| \geq l$, so $\frac{|PS(r)|}{l} \geq 1$, which means $\lfloor \frac{|PS(r)|}{l} \rfloor \geq 1 > 0$. Since $\frac{|EC(v_i)|}{|PS(r)|} > \frac{1}{l}$, we can obtain $\frac{|PS(r)|}{l} < |EC(v_i)|$. Then $\lfloor \frac{|PS(r)|}{l} \rfloor < \frac{|PS(r)|}{l} < |EC(v_i)|$. So we can obtain $0 < \lfloor \frac{|PS(r)|}{l} \rfloor < |EC(v_i)|$. That concludes the proof. □

(2) The selection of each perturbed value of v_i depends on its data type. For numeric data, we can choose the perturbed value v_i' from an ϵ-neighborhood of v_i, namely $[v_i - \epsilon, v_i + \epsilon]$ where ϵ is a small positive number, to avoid significant information loss. We ensure the perturbed value v_i' is different from all existing sensitive attribute values to avoid increasing the frequency of existing sensitive attribute values, so that the perturbation will not cause the frequency of existing sensitive attributes larger than $\frac{|PS(r)|}{l}$.

In summary, the sensitive attribute values in the published dataset may need to be modified (perturbed) to satisfy the l-diversity. For clarity of presentation, we focus on the basic (k,P)-anonymity model without considering the l-diversity in the main text.

5.5 Discussion

We now discuss the guidelines for setting parameter k and P. As mentioned in Section 4.2.1, P must be greater than $1/thr$ to satisfy the privacy requirement, but should be as small as possible to guarantee the utility of the published data. However, the optimal k selection has always been an open problem. To date only one paper [5] has addressed this problem, and the solution was too restrictive to be applied in our problem context.

Meanwhile, it can be seen that (k,P)-anonymity is actually a generalization of k-anonymity. On one hand, if we set P to 1 and remove all PRs from T^*, the results will be same as conventional k-anonymity based on instant value. On the other hand, if we set $P = k$, the results will be an enhanced version of conventional k-anonymity based on pattern similarity. In general, P must be no greater than k.

6 Supporting Customized Data Publishing

So far we have assumed full data publishing (FDP) of both AE and PR from the entire QI attribute set QI. However, a data owner may wish to customize the publishing process. For example, one can tailor the AE by removing the value ranges of a certain part of the QI attributes while releasing the whole PR, or on the contrary release the whole AE in the meanwhile hiding parts of the PR. This section discusses the techniques for supporting queries on such tailored data. Note that to use customized data, the user has to *estimate* the missing value ranges or PR from the published data. In the following, we firstly define the CDP model, and then discuss the estimation techniques.

6.1 The CDP model

Generally, the customized publishing can be defined as a generalization of Definition 3.

Definition 6: *(Customized Data Publishing)* Given QI attributes $QI = \{A_1, \ldots, A_n\}$ in the original table T, each record r^* in the published database T^* generally contains three parts,

- the *QI value ranges* $R_1, \ldots, R_{n1}$, which are generalized from a subset of QI, denoted by $QI_{ae} = \{A_{i_1}, \ldots, A_{i_{n1}}\}$,
- the *QI pattern representation* $PR[r]$, obtained from another subset of QI, denoted by $QI_{pr} = \{A_{j_1}, \ldots, A_{j_{n2}}\}$,
- the *sensitive information* A_S, which is same as its counterpart in T.

It must be noted that the customization happens only *after* the (k,P)-anonymization. That means the k-groups and P-subgroups are still formed on the whole QI attribute set QI. Since both QI_{ae} and QI_{pr} are subsets of QI, it is easy to see that the customized data still conform to the (k,P)-anonymity.

Now let us consider the problem of estimating the missing value ranges or PR to support queries on them. Without loss of the generality, we assume $QI_{ae} \bigcup QI_{pr} = QI$. Regarding the overlap between QI_{ae} and QI_{pr}, there are four possible cases:

1) Equal ($QI = QI_{ae} = QI_{pr}$), which has been addressed in the previous sections;
2) Contained (either $QI_{ae} \subset QI_{pr}$ or $QI_{pr} \subset QI_{ae}$);
3) Overlap ($QI_{ae} \bigcap QI_{pr} \neq \phi$ and not Contained);
4) Disjoint ($QI_{ae} \bigcap QI_{pr} = \phi$).

In the following, we will first consider the estimation for the Contained case (Section 6.2 for $QI_{ae} \subset QI_{pr}$, where missing value ranges have to be estimated, and Section 6.3 for $QI_{pr} \subset QI_{ae}$, where PR has to be estimated). Then we extend the estimation to the Overlap case. However, our estimation techniques cannot handle the Disjoint case.

6.2 Estimation of AE

We will take SAX as an example to illustrate the estimation of the missing value ranges (in AE) for the case $QI_{ae} \subset QI_{pr}$. In the following, we denote by r_i the actual value of r on attribute A_i, and by r_i^- (r_i^+) its lower (upper) bound of its respective value range. The estimation method contains two consecutive steps with the second step being optional.

(1) Our method firstly attempts to estimate the missing value ranges of a record from its available ones (those in QI_{ae}). Given a published record r^*, for any SAX alphabet α occurring in the PR of r^*, we define two notations as follows: $LB(\alpha) = \max\{r_j^- | A_j \in QI_{ae} \bigwedge sax_j < \alpha\}$ and $UB(\alpha) = \min\{r_j^+ | A_j \in QI_{ae} \bigwedge sax_j > \alpha\}$. Subsequently, we can easily prove the following Lemma.

Lemma *1:* Any attribute value r_i having a respective SAX alphabet of α must be lower (upper) bounded by $LB(\alpha)$ ($UB(\alpha)$).

Therefore, the missing value range of r_i can be estimated as $(LB(sax_i), UB(sax_i))$. However, these two bounds may not be applicable if for all A_j in QI_{ae}, no sax_j is smaller (greater) than sax_i in the PR of r^*. In such case, we have $LB(sax_i) = -\infty$ ($UB(sax_i) = \infty$). Such attribute has to undergo the second step of the method.

(2) The second step exploits the property of the SAX transformation to further tighten the LB and UB. As described in Section 5.1, when anonymizing the data, the attribute values in one record are normalized to a mean of zero and a standard deviation of 1 before being converted to an alphabetic string. That means, the normalized attribute value should always be in range $(-\sqrt{n}, \sqrt{n})$ where n is the number of QI attributes. These two numbers, $-\sqrt{n}$ and $\sqrt{n}$, can be used to predict the conservative lower and upper bounds for the original attribute value:

Assuming the original attribute value corresponding to α is x, the mean value of the entire sequence as μ and the standard deviation as σ. Then we have $\frac{x-\mu}{\sigma} > -\sqrt{n} \Rightarrow x > \mu - \sqrt{n} \cdot \sigma$ and $\frac{x-\mu}{\sigma} < \sqrt{n} \Rightarrow x < \mu + \sqrt{n} \cdot \sigma$. Thus, finding a conservative lower (upper) bound for x is equivalent to finding the min (max) value of $\mu - \sqrt{n} \cdot \sigma$ ($\mu + \sqrt{n} \cdot \sigma$). Specifically, we need to compute $\min(\mu - \sqrt{n} \cdot \sigma)$ and $\max(\mu + \sqrt{n} \cdot \sigma)$, given the constraints of the breakpoints β_b ($b = 1, \ldots, level - 1$) for *level*, which are specified by a series of inequations as follows:

- For each finite $LB(\alpha_j)$, $\frac{LB(\alpha_j)-\mu}{\sigma} < \beta_{j-1}$,
- For each finite $UB(\alpha_j)$, $\frac{UB(\alpha_j)-\mu}{\sigma} > \beta_j$,

where $\alpha_j \in \Sigma_{level}$. So the problem of finding $\min(\mu - \sqrt{n} \cdot \sigma)$ and $\max(\mu + \sqrt{n} \cdot \sigma)$ can be regarded as a *linear programming* problem subject to the above series of inequations. We use the well known simplex algorithm [4] to solve it. Then interval $(\min(\mu - \sqrt{n} \cdot \sigma), \max(\mu + \sqrt{n} \cdot \sigma))$ will be used to tighten $(LB(\alpha), UB(\alpha))$. Due to space limit, the details of the linear programming is omitted.

6.3 Estimation of PR

In the case of $QI_{ae} \supset QI_{pr}$, we estimate the missing PR as follows:

1) Utilize the bound estimation method described in Section 6.2 to obtain a conservative bound $(LB(\alpha), UB(\alpha))$ for each alphabet $\alpha \in \sum_{level}$, from the already known $AE(R_i)$ and $PR(sax_i)$ of each attribute $A_i \in QI_{pr}$.
2) For each $A_h \in QI_{ae} - QI_{pr}$, compare R_h with all $(LB(\alpha), UB(\alpha))$ where $\alpha \in \sum_{level}$. Find the α whose range overlaps R_h the most (as more overlap implies greater probability that α could be the actual sax_h). Thus the missing SAX alphabet of A_h is estimated as $sax_h = \{\alpha | \arg\max_{\alpha \in \sum_{level}} (LB(\alpha), UB(\alpha)) \bigcap R_h\}$.

6.4 Estimation of both AE and PR

The estimation methods in the above two subsections can be combined to handle the Overlap case, as described in the following:

1) Obtain a conservative bound $(LB(\alpha), UB(\alpha))$ for each alphabet $\alpha \in \sum_{level}$ from the AE and PR of the overlapping attributes $QI_{ae} \bigcap QI_{pr}$;
2) For each attribute $A_h \in QI - QI_{ae}$, suppose its respective $sax_h = \alpha$, then its value range is estimated as the bound $(LB(\alpha), UB(\alpha))$ obtained in Step 1;
3) For each attribute $A_l \in QI - QI_{pr}$, its value range R_l is compared with all the ranges $(LB(\alpha), UB(\alpha))$ obtained in Step 1. The alphabet α_{max} whose range has the maximum overlap with R_l is the estimated alphabet of A_l.

7 Experimental Results

To evaluate the effectiveness and efficiency of our algorithms, we conduct an extensive experimental study on both real and synthetic datasets. All our experiments are conducted on a PC with a Pentium Dual-Core CPU and 2GB main memory running the Microsoft Windows XP.

7.1 Experiment Settings

Datasets: All datasets used in the experiments are obtained from two publicly available numerical datasets, namely the Motion Capture dataset from CMU [1], denoted as *Motion*, the Random Walk dataset from TSDMA [7], denoted as *Walk*. *Motion* and *Walk* are both well-known and popular data sets for time series analysis and widely used in many papers. For the default *Motion* dataset, we use a part of CMU motion capture dataset (from cmuconvert 01_01-01_04.bvh), which has a cardinality of 20352. The other *Motion* datasets have cardinality from 10000 to 100000 by extracting different number of records from CMU motion dataset for the test of scalability. For all the time-series in *Motion*, 10 attributes are randomly selected as the QI attributes, and one is chosen to be sensitive. As the original *Walk* dataset is small, we split the time-series into segments of length 11, and thereby obtain a *Walk* dataset containing 6553 time-series, each also containing 10 QI attributes and 1 sensitive attribute. All the values in the datasets are normalized to [0,1].

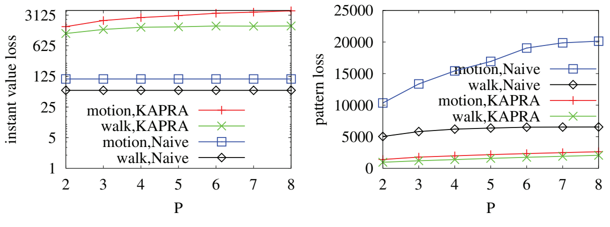
To study the performance of our algorithms on alphabetical sequence data, we also convert the default *Motion* dataset to a string dataset of the same cardinality by SAX using *level*=20. The latter one is referred to as *String*.

To test the scalability thoroughly, we also create a synthetic dataset called *Uniform* containing up to 100000 time-series. Each time-series in this dataset contains 10 attribute values uniformly distributed in range [0,1].

Experiments: The experiments can be divided into three main parts. In the first part, we study the effects of parameter k and P (on the information loss) in the proposed algorithms. In the second part, we compare the utility of the published data between our proposed algorithms (Naive and KAPRA) and two previous works [17][20]. We refer to our implementation of [17] as TGA and that of [20] as BFP2KA. We also study the scalability of all four algorithms in this part. The third part reports our experimental results of Customized Data Publishing.

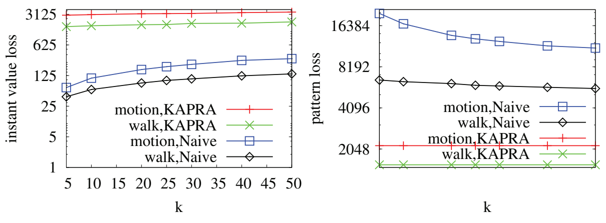
Queries and Metrics: Three types of queries are used and their respective metrics are described as follows:

- **Pattern matching queries**: We use both PSQs and PRQs. (1) For PSQ, the querying time-series is randomly chosen from the database, the tolerance δ is set to λ times the average distance between the pattern vectors extracted from the original and reconstructed series in the database, where λ is a random number in $[1, 10]$; (2) For PRQ, we randomly choose a set of correlation functions in $\mathbf{p}(\cdot)$ as the dimensions for queries. The ranges are obtained using the method described in [11].
  For both query types we use *precision* to measure the results. The precision is defined as
$$precision = |A \bigcap E|/|E|,$$
  where A and E are the result sets obtained from the original and published datasets respectively.



(a) instant value loss (b) pattern loss

Fig. 4. Results of Tuning P (k = 10)



(a) instant value loss (b) pattern loss

Fig. 5. Results of Tuning k (P = 5)

- **Range queries**: Each query contains a number of range predicates on the QI value ranges: *select count(∗) from dataset where* $A_1 \in range_1$ *and* $A_2 \in range_2$ *and ... and* $A_w \in range_w$; The metric is the *relative error* defined as
$$relative\_error = |act - est|/act,$$
  where *act* and *est* are the query results obtained from the original and published dataset respectively.
- **Complex queries**: Each query combines a number of pattern matching predicates (either one similarity predicate or a few range predicates) on the PR and a number of range predicates on the QI value ranges. Similarly, we use the *relative error* to measure the query results.

We also compare the pattern preservation ability of the proposed methods in terms of $PL(\cdot)$. As discussed in Section 4.2.3, $PL(r)$ is given by $distance(\mathbf{p}(r), \mathbf{p}^*(r))$. In the experiments, $\mathbf{p}(r)$ is the vector of all possible differentials between each pair of normalized *QI* attributes of r. The reconstructed vector $\mathbf{p}^*(r)$ is obtained similarly from a numerical sequence r' reconstructed from $PR[r]$. Each value in r' is obtained by the method presented in Section 5.1.

7.2 Tuning k and P

In this subsection, we will show how $VL(\cdot)$ and $PL(\cdot)$ change by varying parameter k and P for both the Naive and KAPRA schemes.

Figure 4 shows the VL and PL values of both Naive and KAPRA by varying P when k = 10. It can be seen that PL continuously increases by P for both schemes. This is expected as when P increases it becomes more difficult to satisfy the P-requirement for both schemes. However, the variation of P has no influence on VL for Naive solution because VL is only determined by k. For KAPRA, VL

slowly increases by P. The results confirm that larger P will cause tremendous information loss. Therefore, on the premise of enough privacy protection ability, P should be as small as possible.

Figure 5 shows the VL and PL by varying k when $P = 5$. The results show that VL increases by k for both Naive and KAPRA. However, the variation of k has no influence on the PL of KAPRA, as the pattern loss is only determined by P. But for the Naive scheme, PL keeps decreasing because a larger k allows for each time-series larger search space to form P-subgroups.

In all the above results, PL of KAPRA is always much lower than that of Naive. This verifies that KAPRA can better preserve the patterns of the original time-series. The VL of KAPRA is higher than Naive. That is because Naive will firstly minimize VL in the group partitioning process.

Another factor worth considering is the portion of suppressed time-series in the whole dataset for KAPRA algorithm. We have mentioned in the above that on the premise of enough privacy protection ability, we will choose a smallest possible number for P as an effort to minimize the pattern loss. We have also asserted in Section 5.3.2 that the number of suppressed time-series will always be less than P. Therefore, the suppressed time-series are generally expected to only occupy a negligible portion of the dataset. Table 4 shows the respective number of suppressed records (line 2-3), mostly zeros, at various P values for *Motion* (20352 records) and *Walk* (6553 records). To illustrate the effectiveness of bad-leaf recycling, we also show on line 4 and 5 of Table 4 the respective number of suppressed time-series without the *recycle bad-leaves* phase (NR). The results indicate that the recycling phase is effective in preserving the time-series contained in the bad-leaves generated from earlier phases.

TABLE 4
Number of Suppressed Time-Series

P	2	5	10	20	30	40	50	100
Motion	0	0	0	0	0	38	0	56
Walk	0	4	0	0	0	0	0	0
Motion(NR)	697	577	608	459	383	525	414	566
Walk(NR)	12	4	0	0	0	0	0	0

7.3 Comparison of Utility and Scalability

In this subsection, we verify the effectiveness of our proposed (k,P)-anonymity by comparing four anonymization schemes for time-series, namely Naive, KAPRA, TGA[17], and BFP2KA[20]. For fairness of comparison, all four schemes are expected to publish data with *identical* breach probability. That means the value of k in TGA and BFP2KA must equal to the value of P in Naive and KAPRA. Therefore, we set $k = P$ for Naive and KAPRA in this set of experiments. We mainly compare the utility of the data published by each scheme. Besides the metrics described in Section 7.1, namely (1) the *precision* of pattern matching query, (2) the *relative error* of range query, (3) the *relative error* of complex query, we also compare (4) the pattern loss (PL) of these schemes. For each query type we generate 10000 random queries and show the average precision or relative error. Finally we evaluate the scalability of all four schemes.

As the first three anonymization schemes handle numerical time-series, we compare Naive, KAPRA, and TGA on numerical datasets (*Motion* and *Walk*). In contrast, as BFP2KA only handles alphabetical sequences, we compare Naive and KAPRA to BFP2KA on the *String* dataset.

Figure 6 shows the results of PL on the *Motion* and *Walk* datasets. We can observe that for both datasets, TGA suffers more severe pattern loss than KAPRA. This confirms the effectiveness of our proposed pattern-preserving anonymization.

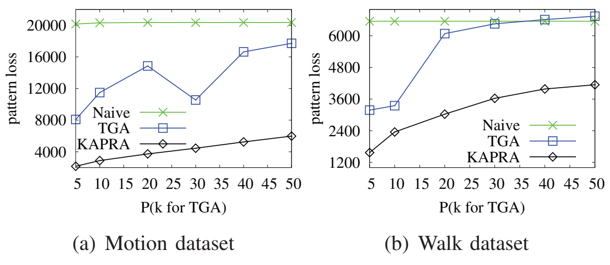


Fig. 6. Comparison of pattern loss

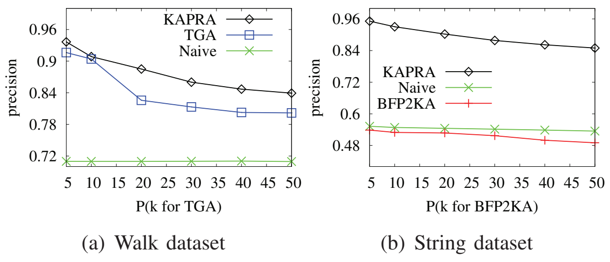


Fig. 7. Results of PSQ

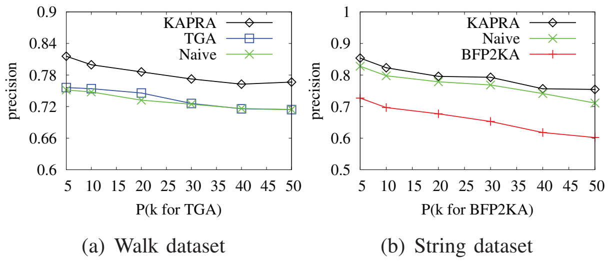


Fig. 8. Results of PRQ

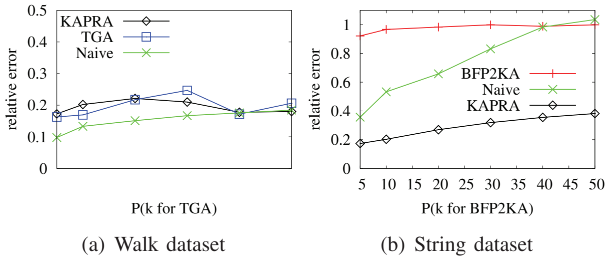


Fig. 9. Results of range queries

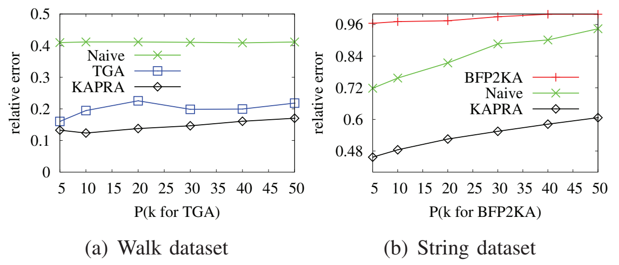


Fig. 10. Results of complex queries

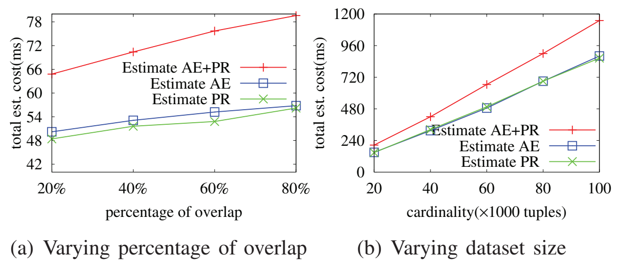


Fig. 12. Computing cost of estimation methods

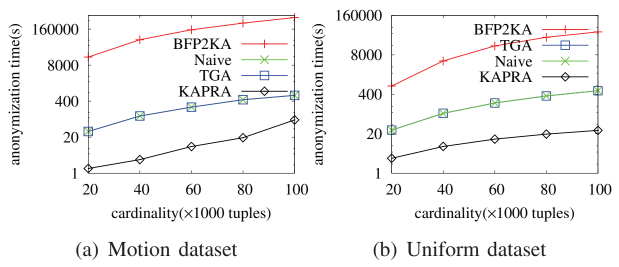


Fig. 11. Scalability

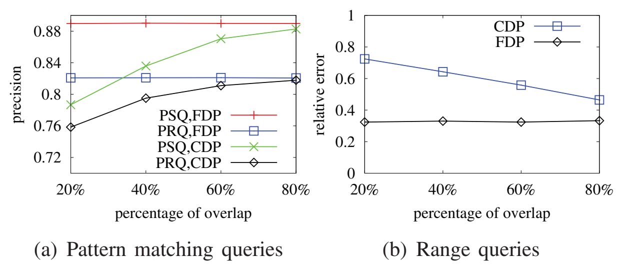


Fig. 13. Utility of customized data publishing

As the query utilities display similar trends on these two datasets, we will only show those on the *Walk* dataset. Figure 7 and Figure 8 show the pattern matching query results on both *Walk* and *String* dataset. We can see that KAPRA considerably outperforms TGA and BFP2KA on both datasets.

Figure 9 shows the results of range queries on both *Walk* and *String*. The Naive scheme slightly outperforms KAPRA and TGA because the Naive scheme will first consider minimizing the instant value loss. KAPRA and TGA achieve comparable performance on this query. But for *String*, KAPRA and Naive scheme considerably outperform BFP2KA.

After that, we compare the relative errors of complex queries in Figure 10. It is obvious that KAPRA outperforms both TGA and BFP2KA in terms of relative errors.

The scalability of all four algorithms are evaluated using the *Motion* and *Uniform* datasets. For BFP2KA, we transform both datasets to alphabets by SAX using *level*=20. For this experiment we set $k = P = 10$. Figure 11 shows the results of the processing time for different dataset sizes. The results for both *Motion* and *Uniform* display similar trends. It can be seen that both the proposed algorithms achieve high efficiency even on very large datasets. KAPRA is the most efficient method while Naive achieves performance comparable with TGA. However, BFP2KA is much slower than the other three schemes.

7.4 Results of Customized Data Publishing

In this subsection we report the results of the proposed CDP techniques. Figure 12(a) shows the total computing cost of the proposed estimation techniques (for AE, PR, and both) for customized publishing of the *Walk* dataset. When varying the *percentage of overlap*, denoted by $op = \frac{|QI_{ae} \bigcap QI_{pr}|}{|QI|}$, the total cost of estimation increases with the percentage of overlap. This is because the increase of overlap will cause higher cost of the $(LB(\alpha), UB(\alpha))$ bound estimation for all the three cases in Section 6. Figure 12(b) shows the total estimation cost on the same series of *Motion* dataset as in Figure 11(a). It can be seen that the estimation methods have good scalability. Since the estimation algorithms are performed once for all before any query processing, their computing costs are apparently trivial.

To evaluate the utility of CDP, we anonymize the *Walk* dataset ($k = 10$ and $P = 3$) using KAPRA, and conduct the CDP as follows: Vary op from 20% to 80%. The rest attributes are evenly distributed between $QI_{ae} - QI_{ae} \bigcap QI_{pr}$ and $QI_{pr} - QI_{ae} \bigcap QI_{pr}$. To reduce the randomness in attribute selection, for each op, we will choose 10 different groups of attributes as the overlapped attributes ($QI_{ae} \bigcap QI_{pr}$) and take the mean value of the query results as the final result. For PRQ and range query, we require all predicates of each query to contain the missing QI attributes. A query load containing 10000 queries is generated for each of the three types of queries. We only present the results of the *Overlap* case because the query results of the two *Contained* cases present similar tendencies.

Figure 13(a) shows the utility of PSQ and PRQ while fixing QI_{pr}, and Figure 13(b) shows that of range queries when fixing QI_{ae}. Both figures show performance improvements (either precision improvement or error reduction) when op increases. When $op = 80\%$, the utilities of all three types of queries on CDP are fairly good compared to FDP. The above results indicate the effectiveness of our estimation methods in supporting CDP.

8 Conclusion

We proposed a novel anonymity model called (k,P)-anonymity for time-series data. Relying on a generic definition to pattern representations, our model could prevent three types of linkage attacks and effectively support the most widely used queries on the anonymized data. We proposed a naive solution and a more advanced method called KAPRA to enforce (k,P)-anonymity on time-series data. Our approach allowed for customized data publishing and provided estimation methods to support queries on such data. The extensive experiments demonstrated the effectiveness of (k,P)-anonymity in resisting linkage attacks while preserving the pattern information of time-series. The KAPRA algorithm outperformed the naive solution in terms of fidelity in pattern preservation and run-time performance. Our results also illustrated the effectiveness and efficiency of the proposed estimation methods for customized data publishing.

(k,P)-anonymity may lead to a few interesting directions for future study. Our current solution imposes a very strict constraint on PR equality and this may cause serious pattern loss. In the future work we will consider loosing the PR equality condition on the premise of ensuring privacy preservation ability. This strategy may greatly reduce the information loss.

References

[1] Cmu graphics lab motion capture database. In *http : //mocap.cs.cmu.edu/*.

[2] O. Abul, M. Atzori, F. Bonchi, and F. Giannotti. Hiding sequences. In *ICDE Workshops*, pages 147–156, 2007.

[3] C. C. Aggarwal and P. S. Yu. A condensation approach to privacy preserving data mining. In *EDBT*, pages 183–199, 2004.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (2ed)*. MIT press and McGraw-Hill, 2001.

[5] R. Dewri, I. Ray, I. Ray, and D. Whitley. On the optimal selection of k in the k-anonymity problem. In *ICDE*, pages 1364–1366, 2008.

[6] D. Gunopulos and G. Das. Time series similarity measures. In *Tutorial PM-2*, pages 243–307, 2000.

[7] E. Keogh and T. Folias. Ucr time series data mining archive. In *http : //www.cs.ucr.edu/ eamonn/TSDMA/*.

[8] E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD Conference*, pages 151–162, 2001.

[9] E. J. Keogh, K. Chakrabarti, M. J. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowl. Inf. Syst.*, 3(3):263–286, 2001.

[10] E. J. Keogh and M. J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *KDD*, pages 239–243, 1998.

[11] J. Li, Y. Tao, and X. Xiao. Preservation of proximity privacy in publishing numerical sensitive data. In *SIGMOD Conference*, pages 473–486, 2008.

[12] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, pages 106–115, 2007.

[13] J. Lin, E. J. Keogh, S. Lonardi, and B. Y. chi Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD*, pages 2–11, 2003.

[14] L.Sweeney. k-anonymity: privacy protection using generalization and suppression. *International Journal on Uncertainty Fuzziness and Knowledge-based Systems*, 10(5), 2002.

[15] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, page 24, 2006.

[16] N. Mohammed, B. C. M. Fung, and M. Debbabi. Walking in the crowd: anonymizing trajectory data for pattern analysis. In *CIKM*, pages 1441–1444, 2009.

[17] M. E. Nergiz, M. Atzori, and Y. Saygin. Perturbation-driven anonymization of trajectories. In *Technical Report 2007-TR-017, ISTI-CNR*, 2007.

[18] J. Nin and V. Torra. Towards the evaluation of time series protection methods. *Inf. Sci.*, 179(11):1663–1677, 2009.

[19] S. Papadimitriou, F. Li, G. Kollios, and P. S. Yu. Time series compressibility and privacy. In *VLDB*, pages 459–470, 2007.

[20] R. G. Pensa, A. Monreale, F. Pinelli, and D. Pedreschi. Pattern-preserving k-anonymization of sequences and its application to mobility data mining. In *PiLBA*, 2008.

[21] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Trans. Knowl. Data Eng.*, 13(6):1010–1027, 2001.

[22] L. Singh and M. Sayal. Privacy preserving burst detection of distributed time series data using linear transforms. In *CIDM*, pages 646–653, 2007.

[23] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Elsevier, third edition, 2006.

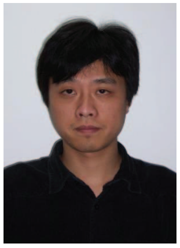
[24] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *VLDB*, pages 139–150, 2006.

[25] J. Xu, W. Wang, J. Pei, and et al. Utility-based anonymization for privacy preservation with less information loss. *SIGKDD Explorations*, 8(2):21–30, 2006.

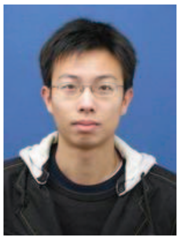
Lidan Shou is associate professor with the College of Computer Science, Zhejiang University, China. He earned his PhD degree in Computer Science from the National University of Singapore. Prior to joining the faculty, he had worked in the software industry for more than two years. His research interests include spatial database, data access methods, visual and multimedia databases, and Web data mining. He is a member of ACM.

Xuan Shang is a Ph.D candidate in the College of Computer Science, Zhejiang University. She earned her BS degree in computer science from Zhejiang University in 2006. Her research interests include data privacy, time series data mining and stream data dissemination.

Ke Chen is Assistant Professor at the College of Computer Science, Zhejiang University. She received her Ph.D. degree in Computer Science in 2007, and later became a post-doctoral fellow at the School of Aeronautics and Astronautics of Zhejiang University until year 2009. Her research interests include spatial temporal data management, Web data mining, and data privacy protection. She is a member of ACM.



Gang Chen is professor with the College of Computer Science and the Director of the Database Lab, Zhejiang University. He earned his PhD degree in Computer Science from Zhejiang University. He has successfully led the investigation in research projects which aim at building China's indigenous database management systems. His research interests range from relational database systems to large-scale data management technologies supporting massive Internet users. He is a member of ACM and senior member of China Computer Federation.



Chao Zhang received his BS degree in computer science from Zhejiang University in 2010. He is currently pursuing M.Sc. degree in the same university. His research interests include spatial databases, indexing techniques, and Web data mining.