PatternKa: Scalable Privacy Algorithm for Sparse High Dimensional Data

ABSTRACT

Data privacy has attracted more and more attention from both consumers and organizations. K-anonymity is a simple, yet effective privacy model proposed to protect data privacy from linkage attacks. In order to overcome the challenge of fast-growing data quantity and dimensionality, we propose a scalable frequent Pattern based K-anonymity algorithm (PatternKa) for anonymizing high dimensional sparse data. PatternKa is the first algorithm adopts frequent pattern mining methods in the data anonymization process and proves the effectiveness of frequent patterns. Experiment results show that PatternKa has superior performance on sparse high dimensional datasets. We show that both set-valued datasets and relational datasets can benefit from PatternKa, which validates the importance of using frequent patterns. We also introduce a distributed version of PatternKa that can run on large-scale datasets with MapReduce framework.

KEYWORDS

k-anonymity, relational data, set-valued data, frequent pattern, scalable, data privacy, high dimensional

ACM Reference Format:

. 2020. PatternKa: Scalable Privacy Algorithm for Sparse High Dimensional Data. In *Proceedings of KDD '20*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Data privacy has attracted lots of interest for decades [17]. In recent years, with privacy laws like General Data Protection Regulation(GDPR) and California Consumer Privacy Act (CCPA) taking into effect, it becomes more and more important for companies to take actions to protect data privacy. K-anonymity model [17] is one of the most popular privacy models. K-anonymity protects privacy by ensuring each record in the dataset to be indistinguishable from another k - 1 records. The advantage of k-anonymity is that it has a simple definition and is easy to verify.

There are other privacy models proposed in recent years to protect data privacy. Differential privacy[4] is one of the popular privacy models proposed to provide statistical information about a dataset while protecting sensitive individual information. Differential privacy methods design ways to control how dataset users can query the dataset. While these methods can successfully postpone the leaking of sensitive information, changing how existing business partners can access the company data may be too costly from a business perspective. That is the reason why simple k-anonymity methods are still applicable to current applications.

KDD '20, August 22-27, 2020, San Diego, CA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: An Example of Set-Valued Dataset

Sample	Item 1	Item 2	Item 3
1	✓		✓
2		✓	
3	✓		

There are many methods proposed to solve the k-anonymity problem [19] [2] [10]. LeFevre et al. [11] proposed a multi-dimensional method that sorts the data along each dimension and partitions across the dimension with the widest normalized range of values, each partition is generalized to a common representation. Since data utility is also an important metric for evaluating k-anonymity results, Xu et al. [20] proposed a data utility measure, normalized certainty penalty (NCP) to evaluate the information loss of both continuous value attribute and categorical value data. Based on the NCP metric, [20] also proposed a top-down method for k-anonymity. Since multi-dimensional k-anonymity problem is proven to be NP-hard [1] [15], [7] proposed an optimal linear time method for one-dimensional (1-D) data anonymization. Based on the efficient 1-D k-anonymity method, [7] also proposed using approximation methods to map multi-dimensional data to 1-D space. These mapping methods partition neighbor data points to close data points in 1-D space with high probability.

K-anonymity is initially proposed to protect data privacy for relational data like hospital patient records and bank client profiles [17]. Besides relational data, there is also set-valued data, such as transaction data, web search queries, and click streams, which consists of a subset of items drawn from a universal itemset. Table 1 shows an example of a set-valued dataset consists of 3 samples. Assuming this dataset contains user purchase records, then sample 1 represents a user has purchased item 1 and item 3. For illustration purpose this example only contains 3 items, while real-world user purchase records usually contain hundreds of items.

While set-valued data poses great challenge for k-anonymity methods for its high dimensionality and large data quantity, some set-valued data has predefined hierarchical relationship between items, this hierarchical relationship can be used to measure similarity between samples. Figure 1 shows an example of a hierarchical tree defined on shopping items. For example, a purchase record that buys detergents is more similar to a purchase record of vacuum cleaners than a purchase record of shoes. Many studies ([18] [9] [6]) use this hierarchical relationship to search for similar items, so that groups after k-anonymity transforms produces less information loss. [18] is the first to propose the k^m-anonymity

model to protect privacy for set-valued data with hierarchical attributes. Along with an optimal solution that searches in attribute hierarchical tree, [18] also proposed two greedy heuristics that scale better and find a solution close to optimal. [9] proposes a top-down, partition-based approach to anonymize hierarchical set-valued data and scores well on an information-loss data quality metric. For sparse high dimensional data anonymization, Ghinita et al. [6] proposed two categories of data reorganization methods. The first uses approximate high dimensional nearest neighbors to form anonymous clusters. The second transforms the data to capture the correlation in the underlying data, then performs an efficient linear time heuristic to anonymize the data. Lin et al. in [13] proposed PTA, which includes three modules, the preprocessing module, the TSP module and the anonymization module. The challenge of high dimensional set-valued data is addressed in the developed system by using a divide-and-conquer approach that partitions the data into several segments based on hamming distance. Their method uses majority voting as the cluster center, which will cause loss of data utility in the sense that a missing transaction (with value 0) may be anonymized to 1, which will change the data truthfulness. For example, it will add non-existent transactions to a user's cart history. This behavior may not always be acceptable in real-world applications.

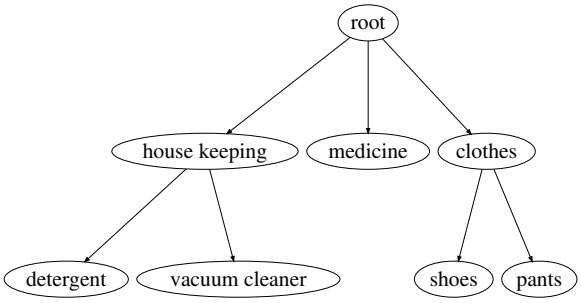

Figure 1: Example of Hierarchical Tree in Shopping Items

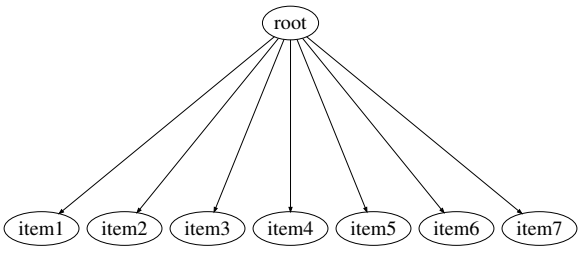

Figure 2: Example of Non-hierarchical Items

Although hierarchical set-valued data is popular among online applications, there may not always exist a hierarchical definition among items. Figure 2 shows an example item relation of non-hierarchical items, the similarity between any items is the same. For example, a purchase record buys item1 has the same similarity to a record that buys item2 or item3. Previous methods on hierarchical set-valued dataset will have poor performance on non-hierarchical set-valued dataset, because there are less information for similarity measurement, and more samples will be similar without hierarchical item relations. In this paper we propose a novel k-anonymity algorithm that is able to anonymize non-hierarchical set-valued dataset. The algorithm can also process large-scale set-valued sparse datasets with a large number of distinct items.

1.1 Motivations

Providing privacy protection for high dimensional data, especially for set-valued data is very important, since set-valued data has many real-world applications. The web search queries, purchase records and click streams are a good source data for data mining research. Without hierarchical relationship between set items brings additional difficulty to the problem. There is a huge amount of non-hierarchical set-valued data in online environment, we will introduce an online advertising scenario to demonstrate the importance of non-hierarchical set-valued data.

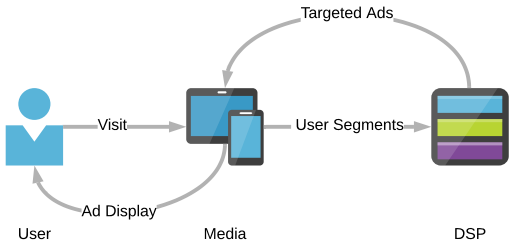

Figure 3: Data Sharing of User Segments

User segment data in audience targeting for online advertising is a good example of set-valued data. Figure 3 shows a common data sharing scenario of user segments. In online advertising industry, where the media show ads to users, since the advertisers want to show the right ad to the right user, detailed user information (user segments) is transferred from media to advertisers or third-party demand side platform (DSP) for audience targeting. In order to select the right user, users are labeled with many non-hierarchical audience targeting labels, called segments, that describes user attributes and past user behaviors. The segment data contains sensitive user privacy information and needs to be anonymized to prevent linkage attacks. Especially, anonymize user segments can prevent attackers to re-identify users based on segments.

Consider a detailed example for segment data privacy, table 2 shows some common user segments used for audience targeting in online advertising. In real-world advertising systems, there can be thousands of user segments with detailed user behavior description, thus it will pose a user privacy threat to directly publish user segments, since an attacker may use a group of segments to re-identify a certain user. More and more users have expressed concerns about the misuse of their online behavior data. Moreover, on

Table 2: Example Advertising Segments

User Id	Female	Male	Age Below 18	Car Owner	Fitness Enthusiast
1		✓		✓	
2		✓	✓		✓
3	✓		✓	✓	✓
4	✓				✓
5		✓			✓
6	✓			✓	✓

the media side, it is common to share a portion of ad inventory to DSP and request for ad display. User segments are transferred between media and third-party DSP for each ad display. Since user segments are required for online audience targeting to achieve better targeting accuracy, it is a balance between user privacy and targeting accuracy. To demonstrate the privacy threat, in the public Netflix movie rating dataset, where each user has a series of movie ratings similar to segments in online advertising, it is shown that most of the user information can be uniquely re-identified [16].

K-anonymity methods can help to protect user data privacy in audience targeting scenarios. Anonymizing user segment data allows both the user and the ad serving companies to protect data privacy on their side, thus preventing further user information leaks, while also provide information for audience targeting. There are little published results on non-hierarchical set-valued data or sparse high dimensional data k-anonymization. Current set-valued k-anonymity methods assume a predefined hierarchical relation between attributes. However, in real-world applications it is hard to acquire attributes hierarchy relation metadata due to ownership and data privacy concerns. For example, in the user targeting scenario, lots of the user profile attributes in ad server come from 3rd-party data manage platforms(DMP). The 3rd-party DMP usually only provide the attribute names rather than attribute hierarchy, which is enough for filtering qualified users. For anonymizing non-hierarchical set-valued data, the previous methods can produce high information loss when performing k-anonymity transforms. This is because most distance metric like euclidean distance and hamming distance under high dimensionality will deteriorate, and produce similar distances between any given data records, which increases the difficulty of grouping similar data records, and high intra-group sample similarity will decrease group information loss.

1.2 Contributions

In this paper we propose the frequent pattern based k-anonymity algorithm (PatternKa), that provides insight for sparse high dimensional data, especially non-hierarchical set-valued data k-anonymity problems and produces superior performance. This method is based on frequent patterns of the set-valued data and is scalable for large-scale high dimensional sparse data, which has many real-world applications.

PatternKa firstly mine all the closed frequent patterns, then propose a greedy heuristic based on longest closed frequent pattern, which achieves state-of-the-art performance on information loss metric. The output of the method is a valid partition of the input dataset. Each partition of the data contains at least k records, each record is anonymized by suppressing attributes other than the common attributes in the same partition.

The anonymized data can be used for data publishing, using a method similar to Anatomy [19]. For example, in the online advertising scenario, we publish the anonymized user segments to the third-party DSPs so that attackers cannot use user segments to re-identify any certain user.

PatternKa addresses the problem of data high dimensionality, by converting original data into the space of closed frequent patterns, which corresponds to the common attributes in each partition. The longest frequent pattern heuristic captures an important influence factor of global information loss of the partitioned data. Our specific contributions are:

- We are the first research combine frequent pattern mining methods with k-anonymity to provide an efficient data privacy algorithm.
- We introduce a novel k-anonymity algorithm that can protect sparse high dimensional data privacy with low information loss. We show that our method clearly outperforms other methods on benchmark datasets.
- The proposed PatternKa algorithm can handle both set-valued data and relational data, this proves PatternKa is a general algorithm for data anonymization.
- We prove that our method is scalable for high dimensional sparse dataset by devising a distributed version of our data anonymization algorithm. The distributed version of the algorithm can run on Apache Spark and evaluation results on large datasets show promising results.
- We introduce an important user data anonymization scenario in online advertising, this shows that anonymizing sparse high dimensional data has many real-world applications.

The rest of the paper is described blow: The background and metrics definition is introduced in section 2. Section 3 introduces the design and implementation of proposed methods, and in section 4 we show our method is scalable and a distributed version of algorithm is described. The experiment results are shown in section 5 on public datasets.

2 BACKGROUND

We firstly introduce our privacy model for anonymizing set-valued data, then introduce relational data can be modified to apply this algorithm.

Consider a data table consisting of n rows (or n samples) as is shown in table 3:

$$T = (t_1, t_2, ..., t_n) \tag{1}$$

Where each sample t_i has M attribute values

$$t_i = (a_{i1}, a_{i2}, ..., a_{iM}) \tag{2}$$

As is shown in table 3 quasi-identifiers is a minimal set of attributes $(A_1, A_2, ..., A_m)$ that can be joined with external information to re-identify individual records, where A_i is the $i-th$ quasi-identifier.

Table 3: Example Dataset

	0	1	2	3	...	M
t_1	a_{10}	a_{11}	a_{12}	a_{13}		a_{1M}
t_2	a_{20}	a_{21}	a_{22}	a_{23}		a_{2M}
t_3	a_{30}	a_{31}	a_{32}	a_{33}		a_{3M}
...						
t_n	a_{n0}	a_{n1}	a_{n2}	a_{n3}		a_{nM}
	A_1		A_2			A_m

In this paper we assume all the attributes in the dataset are quasi-identifiers. For relational dataset, attribute value a_{ij} can have categorical values or continuous values; while for set-valued dataset, each attribute value a_{ij} can only have value 1 or 0, representing whether the attribute is present or not. Compared to [18], for set-valued data we assume the attacker has all the user segment information for a subset of users, this means the attacker can re-identify any certain user given that user's attribute values. Similarly, for relational data, we keep the same assumption.

Formally, given a parameter k and the quasi-identifiers, a dataset T is said k-anonymous if for each sample t ∈ T, there exist at least another (k - 1) samples $(t_1, t_2, ..., t_{k-1})$ such that they share the same attribute value on the quasi-identifiers. We define the set of samples with equal quasi-identifier value as a group. The problem of k-anonymity is to find a method to anonymize a dataset T into T' such that T' is k-anonymous and T' is as close to T as possible according to some quality metric.

2.1 Metrics

In order to measure how well the generated samples approximate the original ones, we adopt Normalized Certainty Penalty (NCP) similar in [20] [7] to be the information loss metric. After some k-anonymity transformation, each sample t_i in the dataset belongs and only belongs to one group g. For numeric valued attributes A_i, the NCP of a group g is defined as:

$$NCP_{A_i}(g) = \frac{max_g(A_i) - min_g(A_i)}{max_T(A_i) - min_T(A_i)} \tag{3}$$

Where the numerator represents the range of attribute A_i in g:

$$\begin{aligned} max_g(A_j) &= max\{a_{ij} | t_i \in g\} \\ min_g(A_j) &= min\{a_{ij} | t_i \in g\} \end{aligned} \tag{4}$$

And the denominator represents the range of attribute A_i in the entire dataset T:

$$\begin{aligned} max_T(A_j) &= max\{a_{ij} | t_i \in T\} \\ min_T(A_j) &= min\{a_{ij} | t_i \in T\} \end{aligned} \tag{5}$$

For categorical attributes A_j, assuming there is a hierarchical relationship on attribute values similar to the relation in figure 1, NCP is defined as:

$$NCP_{A_j}(g) = \begin{cases} 0 & (if\ Leaf(u) = 1) \\ \frac{Leaf(u)}{|A_j|} & (else) \end{cases} \tag{6}$$

Where u is the lowest common ancestor of all A_j values in group g:

$$u := lowest\_common\_ancestor\{a_{ij} | t_i \in g\} \tag{7}$$

$Leaf(u)$ is the total number of leafs in the subtree with root u. For set-valued attributes, different values of A_i share no extra information, therefore NCP is defined as:

$$NCP_{A_k}(g) = \begin{cases} 0 & (a_{ij} = 1\ for\ all\ t_i \in g) \\ 1 & (else) \end{cases}$$

The NCP of a group g over all quasi-identifiers is:

$$NCP(g) = \sum_{i=1}^{M} |g| \cdot NCP_{A_i}(g) \tag{8}$$

Where M is the number of distinct attributes in the dataset, $|g|$ is the number of rows in g. In order to measure the information loss on the entire dataset T' after k-anonymity operation, the Global Certainty Penalty(GCP) defined on the whole dataset is:

$$GCP(T') = \frac{\sum_{g \in T'} |g| \cdot NCP(g)}{\Omega} \tag{9}$$

Where Ω is the total information of the input dataset. For relational dataset, Ω is defined as:

$$\Omega = M * N \tag{10}$$

Where M is the total number of attributes in each record, and N is the total number of records in the dataset. For set-valued data, Ω is defined as:

$$\Omega = \sum_{i=1}^{N} \sum_{j=1}^{M} a_{ij} \tag{11}$$

Where a_{ij} has value 1 if t_i contains attribute A_j, 0 otherwise. We use GCP to evaluate the performance of our method against other baseline methods.

3 FREQUENT PATTERN GREEDY METHOD

In this section, we introduce our k-anonymity method – scalable frequent Pattern based K-anonymity algorithm (PatternKa) on set-valued data. Moreover, we show that with little modification, our method can also be applied on relational data.

3.1 Set-valued Data Anonymization

For set-valued data, each sample t_i has a set of attributes $(a_{i1}, a_{i2}, ..., a_{im})$, where each attribute is either 0 or 1. After k-anonymity operation, the s resulting groups $g_1, g_2, ..., g_s$ will have at least k records each. Given group g_i, the center of group g_i is defined as:

$$C_i = \bigcap_{j=1}^{|g_i|} t_j = \bigcap_{j=1}^{|g_i|} (a_{j1}, a_{j2}, ..., a_{jm}) = (c_{i1}, c_{i2}, ..., c_{im}) \tag{12}$$

Where t_j is a record in group g_i, $|g_i|$ is the total number of records in g_i, and $(a_{j1}, a_{j2}, ..., a_{jm})$ is the binary attribute in t_j. The ∩ operator between two records is similar to set intersection:

$$t_i \cap t_j = (a_{i1} \wedge a_{j1}, a_{i2} \wedge a_{j1}, ..., a_{im} \wedge a_{jm}) \tag{13}$$

Where ∧ is the boolean AND operator.

As shown in formula 12, the center of a group g_i is a subset of attributes that has value 1 in all the records in g_i. This leads us to a claim:

C.1 Attributes A_j in C_i with value $c_{ij} = 1$ forms a frequent pattern of the original set-valued dataset with frequency threshold k.

The proof of claim **C.1** is straight forward since $|g| \geq k$ for any group g, therefore each attribute A_j in C_i with $c_{ij} = 1$ is a frequent item. We will show later this is an important property that can lead us to an efficient k-anonymity method.

Firstly, we define Result Information (RI) as the total number of set-valued attributes remaining in the result dataset:

$$RI(T') = \sum_{i=1}^{s} \eta(C_i) \cdot |g_i| \tag{14}$$

Where $\eta(C_i)$ is the number of non-zero elements in group center C_i, and also represents the length of frequent pattern in g_i.

$$\eta(C_i) = \sum_{j=1}^{m} c_{ij} \tag{15}$$

According to defination 14, RI corresponds to the number of sample attributes that remains after the k-anonymity transformation. From 15 we can infer that RI is a weighted sum of frequent pattern lengths of each group g_i.

RI has a direct relationship with information loss GCP:

$$GCP(T') = \frac{\Omega - RI(T')}{\Omega} = 1 - \frac{RI(T')}{\Omega} \tag{16}$$

Where Ω is the total information in the input dataset. Formula 16 lead us to an important claim:

C.2 Larger weighted sum of g_i frequent pattern lengths lead to less information loss.

The correctness of this claim can be shown from formula 14, 16:

$$GCP(T') = 1 - \frac{\eta(C_i) \cdot |g_i|}{\Omega} = 1 - \omega \cdot \eta(C_i) \tag{17}$$

Where $\omega = \frac{|g_i|}{\Omega}$ is the weight term, and C_i is a frequent pattern according to **C.1**.

Based on claim **C.2**, in order to have longer frequent patterns in the grouping results, we propose the PatternKa algorithm, as is shown in algorithm 1. PatternKa is an iterative algorithm, based on remaining data, it produces anonymized data groups one at a time. We mine all the frequent patterns for a given dataset, then as a heuristic, choosing the longest frequent pattern p_m at each iteration to form the next group.

For set-valued data, the *PatternSelection* process is randomly select a pattern from input frequent patterns Π_{max}. The *SampleSelection* process is:

$$C_p = \{t \mid t \in T, p \subseteq t\} \tag{18}$$

The main complexity of this algorithm lies within the frequent pattern mining process. In order to efficiently compute the group centers, we compute all the frequent patterns only once before choosing each group center and validate the pattern frequency at each iteration. There are also many other improvements, which will be covered in section 3.3.

Algorithm 1: PatternKa Clustering Algorithm

Input:
Input dataset T
Grouping parameter K
Output:
Grouped Data $\{g_1, g_2, g_3, ..., g_m\}$

Let clusters $\Phi = \emptyset$
while $|T| >= K$ **do**
  // Choose longest patterns;
  Π = Frequent Pattern(T);
  $l_{max} = max\{length(p) \mid p \in \Pi\}$;
  $\Pi_{max} = \{p \mid length(p) = l_{max}, p \in \Pi\}$;
  $p_m = PatternSelection(\Pi_{max})$;

  // Form a new group;
  $C_p = SampleSelection(p, T)$;
  Append C_p to Φ as a new group;
  Remove C_p from T;
end
Find group C_m in Φ that has minimum NCP;
Assign remaining rows in T to C_m;
output clusters Φ;

Table 4: Continuous Attribute Buckets

Bucket Index	**1**	**2**	**...**	**B**
Age	0-18	18-36	...	72+

3.2 Relational Data Anonymization

In this section we show that with little modification PatternKa can be applied on relational data. Compared to set-valued data, relational data has categorical and continuous attributes. This means that it is hard to form frequent patterns since there will be more possible values in categorical and continuous attributes.

According to definition 3 and definition 6, a smaller information loss needs the attribute values to be more similar within the same group. Therefore, we process categorical attributes and continuous attributes separately and convert them to set-valued data, so that we can apply algorithm 1. For categorical attributes, we relabel the attribute values by the in-order traversal of the attribute hierarchical tree, such that grouping attributes values close to each other will produce smaller information loss. For continuous attributes, we firstly divide the value range of the continuous attributes into B fix length buckets, then relabel the continuous attributes values by the bucket index. Table 4 shows an example of age buckets.

After indexing both categorical and continuous attributes, we can convert the original relational dataset to a set-valued dataset with indexes as attributes. This bucketing algorithm is shown in algorithm 3.

PatternKa for relational data firstly converts relational dataset to set-valued dataset using algorithm 3. Then replace the *PatternSelection* and *SampleSelection* process in algorithm 1.

Algorithm 2: *PatternSelection − Relational*

Input:
The initial data T
Frequent patterns Π
Output:
Pattern p

Let group set $\Phi = \emptyset$;
for *pattern p in* Π **do**
  $C_p^{raw} = \{t \mid t \in T, p \subseteq t\}$;
  Find k-th smallest diversity score $\zeta^{(k)}(C_p^{raw})$ in C_p^{raw};
  $C_p = \{t \mid t \in C_p^{raw}, \zeta(t, C_p^{raw}) \leq \zeta^{(k)}(C_p^{raw})\}$;
  Add C_p to Φ;
end
$n_{min} = min\{NCP(C_p) \mid C_p \in \Phi\}$;
$P_{min} = \{p \mid NCP(C_p) = n_{min}, C_p \in \Phi\}$;
$p = RandomSample(P_{min})$;
Output p;

The *PatternSelection* process for relational data depends on score $NCP_{pattern}(p)$:

$$NCP_{pattern}(p) = NCP(C_p^{raw}) \tag{19}$$

Where C_p^{raw} is all the samples in dataset T containing pattern p:

$$C_p^{raw} = \{t \mid t \in T, p \subseteq t\} \tag{20}$$

The pattern with minimum score is selected. Since the number of samples contains pattern p may be very large, we use diversity score ζ to select a subset of samples C_p from C_p^{raw}. The diversity score metric $\zeta(t_i, C_p^{raw})$ measures the potential diversity increase to include a sample t_i in the final group C_p:

$$\zeta(t_i, C_p^{raw}) = \prod_{j=1}^{m} freq(a_{ij}, C_p^{raw}) \tag{21}$$

Where $freq(a_{ij}, C_p^{raw})$ is the frequency of attribute value a_{ij} in C_p^{raw}:

$$freq(a_{ij}, C_p^{raw}) = |\{v | v = a_{ij}, v \in C_p^{raw}\}| \tag{22}$$

After deriving diversity score of all samples in C_p^{raw}, the k-th smallest sample diversity score $\zeta^{(k)}(C_p^{raw})$ is used as score threshold to filter top k smallest diversity score samples in C_p^{raw}:

$$C_p = \{t \mid \zeta(t, C_p^{raw}) \leq \zeta^{(k)}(C_p^{raw})\} \tag{23}$$

The output pattern is the pattern with minimum $NCP(C_p)$ score according to equation 8. This *PatternSelection* process is shown in algorithm 2.

The *SampleSelection* process for relational data is simply output C_p from algorithm 2.

3.3 Runtime Performance Improvements

We propose several effective techniques that can significantly speed up the PatternKa algorithm 1 . The most time-consuming subroutine in the PatternKa is the frequent pattern mining process,

Algorithm 3: Bucket Attributes

Input:
The initial data T
Split bucket number B
Output:
Bucketed dataset T_{bucket}

$T_{bucket} = \emptyset$;
Init hash-map $\mathbb{M}$;
for *attribute* A_i *in* T **do**
  if A_i *is categorical* **then**
    for *attribute value* a_i *in* A_i **do**
      $inorder(a_i) \leftarrow$ the in-order traversal index of a_i;
      $\mathbb{M} = \mathbb{M} \bigcup \{a_i \rightarrow inorder(a_i)\}$;
    end
  else
    // A_i is continuous
    $A_{max} = max(T(A_i))$;
    $A_{min} = min(T(A_i))$;
    $R_{A_i} = A_{max} - A_{min}$;
    $step = \frac{R_{A_i}}{B}$;
    for *attribute value* a_i *in* A_i **do**
      $I_{a_i} = \lfloor \frac{a_i}{step} \rfloor$;
      $\mathbb{M} = \mathbb{M} \bigcup \{a_i \rightarrow I_{a_i}\}$;
    end
  end
end
for t *in* T **do**
  $t_{bucket} = \{\mathbb{M}(a) \mid a \in t\}$;
  Add t_{bucket} to T_{bucket};
end
Output clusters T_{bucket}

in this section we will introduce four performance improvements for the PatternKa algorithm:

- Closed frequent pattern mining
- Pattern length pruning
- Longest closed pattern caching
- Bitset pattern counting

The first improvement is closed frequent pattern mining. The closed frequent pattern and maximum frequent pattern definition is similar to [14], as is shown in section 7.1. In order to find longest frequent patterns, PatternKa mine closed frequent patterns or maximum frequent patterns, instead of all frequent patterns. This method can reduce the runtime complexity of frequent pattern mining algorithm.

Another improvement is pattern length pruning. This method can be applied on many frequent pattern mining algorithms([8] [14] [21]). Take DCI_CLOSED algorithm [14] [5] as an example, in the main loop of DCI_CLOSED procedure, a validation function is called to determine whether it is possible to find longer frequent patterns compared to the previous found frequent pattern. For example, in DCI_CLOSED algorithm, if the

combined length of CLOSED_SET and POST_SET is less than the previous longest frequent pattern length l_{prev}:

$$|CLOSED\_SET \bigcup POST\_SET| \leq l_{prev} \tag{24}$$

Then there is no need to keep searching for longer patterns. This is an important improvement since it can greatly reduce the total amount of frequent patterns mined.

The third improvement is longest closed pattern caching. As is proved in section 7.2, since there cannot be any new closed frequent patterns found in the same dataset, this method caches closed frequent patterns after the frequent pattern mining process, then counting the frequency of each pattern to determine whether it is still a frequent pattern, as is shown in algorithm 4. Since there may exist many closed frequent patterns, we only cache the longest closed frequent patterns to reduce memory cost. Longest closed pattern caching effectively reduces the number of frequent pattern mining calls.

Algorithm 4: Mining Longest Closed Frequent Patterns with Caching

Input:
Set-valued data D
Anonymization parameter K
Cached closed frequent patterns Π_{cached}
Output:
Longest closed frequent patterns

f_Π = count frequency of Π_{cached} in D;
Π_{cached} = filter Π_{cached} by f_Π > K;
if Π_{cached} *is empty* **then**
  $\Pi_{closed} = ClosedFrequentPatterns(D)$;
  $l_{max} = max\{|p| \mid p \in \Pi_{closed}\}$;
  $\Pi_{cached} = \{p \mid p \in \Pi_{closed}, |p| = l_{max}\}$;
end
output Π_{cached};

C.3 The longest frequent patterns used in PatternKa algorithm iterations is a subset of closed frequent patterns in original input dataset.

The fourth improvement is bitset pattern frequency counting. Even with longest closed frequent patterns cached, it may still be time-consuming to recount the frequency of each pattern. This paper proposes a bitset frequency counting method that can significantly speed up the process in practice. Our method is based on the DCI_CLOSED implementation [5], where we modify the output of the algorithm to produce frequent patterns along with pattern bitset masks in the dataset. The bitset mask $\mathbb{M}_p$ of a pattern p has length equal to the dataset size N, where each bit is either 1, represents the pattern is contained in the corresponding data sample, or 0 otherwise. Figure 4 shows an example of pattern bitset mask.

The frequency of a pattern p is the number of non-zero elements in $\mathbb{M}_p$. $\mathbb{M}_p$ needs to be updated along with the dataset. We note the data bitset mask for the removed data C_p in algorithm 1 as $\mathbb{M}_g$, then the average runtime complexity of update $\mathbb{M}_p$ is:

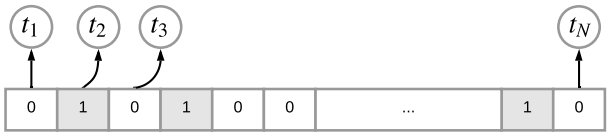

Figure 4: Pattern Bitset Mask Example

$$O(|\mathbb{M}_p \cap \mathbb{M}_g|) = O(K) \tag{25}$$

In comparison, the complexity of naively counting pattern frequency is $O(M \cdot N)$, assuming the pattern length is M, the dataset size is N. This shows bitset pattern frequency counting can significantly improve the runtime complexity.

4 DISTRIBUTED K-ANONYMITY

In this section we introduce the distributed version of our algorithm as is shown in algorithm 5. The distributed version of PatternKa algorithm is implemented on Apache Spark with MapReduce framework.

Algorithm 5: Distributed Longest Closed Frequent Pattern Mining

Input:
Distributed Set-valued data D
Anonymization parameter K
Output:
Longest Frequent Patterns Π

// Mapper
let key value map $M_{kv} = \emptyset$;
for *t in D* **do**
  m = |t|;
  for *index j from 1 to m* **do**
    add key value pair (t_j, t[1:j]) to M_{kv};
  end
end
// Reducer
Aggregate values in M_{kv} by key;
Let frequent pattern set $\Pi = \emptyset$;
for *key* κ, *partition* D_κ *in* M_{kv} **do**
  $\Pi_{closed} = ClosedFrequentPatterns(D_\kappa)$;
  $l_{max} = max\{|p| \mid p \in \Pi_{closed}\}$;
  $\Pi_{max} = \{p \mid p \in \Pi_{closed}, |p| = l_{max}, LastElement(p) = \kappa\}$;
  $p_l = RandomSample(\Pi_{max})$;
  Add p_l to Π;
end
Output Π;

Similar to local PatternKa algorithm 1, the main procedure in calculating anonymity groups is to find longest closed frequent patterns. In this paper, we adopt a distributed closed frequent pattern mining method, similar to Pfp proposed in [12].

Table 5: BMS-WebView Dataset Statistics

Dataset	Dataset Size	Distinct Items	Maximum Row Size	Average Row Size
BMS-WebView-1	59,602	497	267	2.5
BMS-WebView-2	77,512	3,340	161	5.0

The distributed longest closed frequent pattern mining algorithm is shown in algorithm 5. The algorithm has two stages: mapping stage and reduce stage. In the mapping stage, a mapper is used to distribute data into different partitions, which may exist on different machines. This process produces a set of conditional itemset with the last item as the key. The dataset is then distributed according to the key of the conditional itemset. In the reduce stage, the key-value pair produced by the mapper is aggregated by the key. Frequent patterns for each partition are mined by some frequent pattern mining algorithms, for example in [12], the authors use FP-Growth tree structure to mine frequent patterns. In this work, we use a modified DCI_CLOSED data structure [14] to mine longest closed frequent patterns. The longest closed frequent patterns for each key κ is then simply aggregated to produce the final longest closed frequent pattern set. The correctness of the algorithm is shown in [12].

After deriving the longest frequent pattern, we form new data groups similar to algorithm 1.

5 EXPERIMENTS

In this section we evaluate our method on public datasets against the existing methods and show that our method has superior performance on both information loss metric and runtime metric. All algorithms were run on Intel i7 2.2GHz machine with 16GB RAM and macOS version 10.14.6.

Our experiment dataset consists of the Adult Dataset, BMS-WebView dataset and Netflix dataset.

The Adults census dataset from the UC Irvine Machine Learning Repository has become the benchmark dataset for k-anonymity. The dataset is described in [3], after removing missing values, the resulting data contains 30,162 rows and 9 columns. Among remaining columns, there is one continuous value column age, along with 8 categorical columns: employ_status, education, marital, occupation, race, sex, nation and income.

BMS-WebView-1 and BMS-WebView-2 are set-valued dataset and are introduced in [22]. They contain real-world item click data from two e-commerce web sites, collected over a period of several months. Each sample of the BMS-WebView dataset is a transaction record. A transaction is a set of items with different sizes. The statistics of BMS-WebView-1 and BMS-WebView-2 dataset have been displayed in table 5.

Netflix released a movie rating dataset to improve movie recommendation algorithms(New York Times, October 2 2006). As shown in [16], using little prior knowledge about the movie rating history, 96% of the subscribers can be uniquely re-identified. We transform the netflix dataset, by collecting all the movie a user has rated, to form a set-valued dataset for each user. The result set-valued dataset is similar to BMS-WebView dataset, in that the movie list for a user is corresponding to the clicked items in online e-commerce web history. For the transformed netflix dataset, there are 480,189 subscribers in total, 6,733 distinct items with an average of 56 movie ratings for each user.

5.1 Experiment on Adult Dataset

In order to evaluate our k-anonymity algorithm on relational data, we show our results on Adult dataset and compare with previous methods including mondrian [11] and hilbert space filling curve method [7].

The Adult dataset is a relational dataset, we convert the dataset to set-valued dataset using algorithm 3. The age feature is bucketed with parameter B = 20. The NCP and execution time metric result is shown in figure 5. Our method shows superior NCP in different values of K. The execution time of the proposed algorithm is comparable to the mondrian method, while the hilbert space filling curve [7] method is faster for small values of k, since the method uses a dynamic programming algorithm on 1-D data, the execution time of hilbert space filling curve method will grow with the value of k.

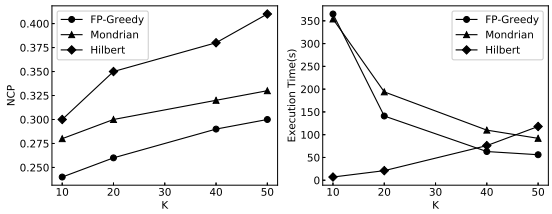

Figure 5: NCP on Adult Dataset

5.2 Experiment on BMS-WebView Dataset

In order to evaluate our method on set-valued data, we show our result on the BMS-WebView-1 dataset. We assume there is no hierarchical information among the items in the dataset. The NCP metric and execution time is shown in figure 6.

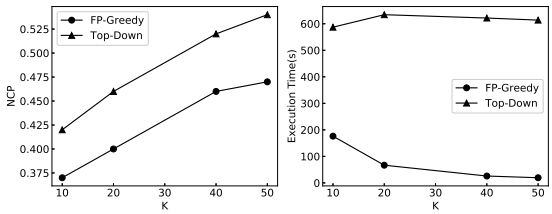

Figure 6: NCP on BMS-WebView-1 Dataset

For BMS-WebView-2 dataset, since the number of items increased 10 times compared to BMS-WebView-1 dataset, the top-down algorithm cannot finish execution in a reasonable time. This

Table 6: Netflix combined_data_1 Dataset Statistics

Dataset	Dataset Size	Distinct Items	Maximum Row Size	Average Row Size
netflix	410,201	4,496	2,060	13.4

Table 7: NCP and Execution Time on Netflix Dataset

K	Machine Number	NCP	Execution Time(hours)	Cluster Number
500	80	0.838	3	687
100	80	0.798	14	318

is because in the non-hierarchical scenario, each split in the top-down algorithm has to consider every possible split item and calculate its NCP metric after the split, this process is very time-consuming. For the above reason, we only show PatternKa algorithm result on dataset BMS-WebView-2 in figure 7.

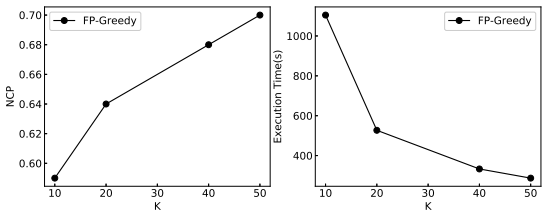

Figure 7: NCP on BMS-WebView-2 Dataset

5.3 Experiment on Netflix Dataset

The netflix dataset is originally used for training prediction models that predict user ratings for films. Since the dataset is large compared to other benchmark datasets in k-anonymity research, we transform the dataset into a large set-valued dataset in order to show that our method can anonymize large dataset with superior NCP metric. The netflix dataset consists of four file parts, we show our result on the first part, namely the combined_data_1.txt file. Other file parts have similar results. The statistics of the set-valued data is shown in table 6.

We use the distributed version of PatternKa algorithm to anonymize the netflix dataset. The distributed PatternKa algorithm is implemented with Apache Spark and runs on yarn cluster with 80 machines. Each machine has 2 cores, 12 GB memory. The NCP metric result and execution time result is shown in table 7. As is shown in the table, our method can finish in a reasonable time, where other methods cannot process such large data size with comparable NCP.

6 CONCLUSION

In this paper we propose a novel sparse high dimensional data k-anonymize algorithm that links frequent patterns with k-anonymity clusters. The PatternKa algorithm captures the key challenge of k-anonymity problem, and a heuristic based on this relation shows high improvement compared to previous methods.

As distributed computing plays an important role in real-world applications, we also extend our method to use the MapReduce framework for large datasets, making our method applicable to real-world applications. With the advances in frequent pattern mining research, we believe our method can adopt advances in frequent pattern mining methods to achieve superior results.

REFERENCES

[1] Charu C Aggarwal. 2005. On k-anonymity and the curse of dimensionality. In *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 901–909.
[2] Chalasani Ajun. [n.d.]. Slicing: A New Approach to Privacy Preserving Data Publishing. ([n. d.]).
[3] Roberto J Bayardo and Rakesh Agrawal. 2005. Data privacy through optimal k-anonymization. In *21st International conference on data engineering (ICDE'05)*. IEEE, 217–228.
[4] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.
[5] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Zhihong Deng, and Hoang Thanh Lam. 2016. The SPMF open-source data mining library version 2. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 36–40.
[6] Gabriel Ghinita, Panos Kalnis, and Yufei Tao. 2010. Anonymous publication of sensitive transactional data. *IEEE Transactions on Knowledge and Data Engineering* 23, 2 (2010), 161–174.
[7] Gabriel Ghinita, Panagiotis Karras, Panos Kalnis, and Nikos Mamoulis. 2007. Fast data anonymization with low information loss. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 758–769.
[8] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. In *ACM sigmod record*, Vol. 29. ACM, 1–12.
[9] Yeye He and Jeffrey F Naughton. 2009. Anonymization of set-valued data via top-down, local generalization. *Proceedings of the VLDB Endowment* 2, 1 (2009), 934–945.
[10] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. 2005. Incognito: Efficient full-domain k-anonymity. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM, 49–60.
[11] Kristen LeFevre, David J DeWitt, Raghu Ramakrishnan, et al. 2006. Mondrian multidimensional k-anonymity.. In *ICDE*, Vol. 6. 25.
[12] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y Chang. 2008. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems*. ACM, 107–114.
[13] Jerry Chun-Wei Lin, Qiankun Liu, Philippe Fournier-Viger, and Tzung-Pei Hong. 2016. PTA: An efficient system for transaction database anonymization. *IEEE Access* 4 (2016), 6467–6479.
[14] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. 2005. Fast and memory efficient mining of frequent closed itemsets. *IEEE Transactions on Knowledge and Data Engineering* 18, 1 (2005), 21–36.
[15] Adam Meyerson and Ryan Williams. 2004. On the complexity of optimal k-anonymity. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 223–228.
[16] Arvind Narayanan and Vitaly Shmatikov. 2006. How to break anonymity of the netflix prize dataset. *arXiv preprint cs/0610105* (2006).
[17] Latanya Sweeney. 2002. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 05 (2002), 557–570.
[18] Manolis Terrovitis, Nikos Mamoulis, and Panos Kalnis. 2008. Privacy-preserving anonymization of set-valued data. *Proceedings of the VLDB Endowment* 1, 1 (2008), 115–125.
[19] Xiaokui Xiao and Yufei Tao. 2006. Anatomy: Simple and effective privacy preservation. In *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 139–150.
[20] Jian Xu, Wei Wang, Jian Pei, Xiaoyuan Wang, Baile Shi, and Ada Wai-Chee Fu. 2006. Utility-based anonymization using local recoding. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 785–790.
[21] Mohammed J Zaki and Ching-Jui Hsiao. 2002. CHARM: An efficient algorithm for closed itemset mining. In *Proceedings of the 2002 SIAM international conference on data mining*. SIAM, 457–473.
[22] Zijian Zheng, Ron Kohavi, and Llew Mason. 2001. Real world performance of association rule algorithms. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 401–406.

7 APPENDIX

7.1 Frequent Pattern Definitions

In this section we give the definition of closed frequent patterns and maximum frequent patterns. For a sample set $R \subseteq T$ and itemset $I \subseteq \mathcal{I}$, where $\mathcal{I}$ is the finite set of items in T, we define two functions f and g:

$$f(R) = \{i \in \mathcal{I} | \forall r \in R, i \in r\} \tag{26}$$

$$g(I) = \{r \in T | \forall i \in I, i \in r\} \tag{27}$$

Where $f(R)$ returns the set of items contained in R, and $g(I)$ returns all rows containing I. A closed itemset I, or closed pattern I, is defined based on f and g:

$$I = f(g(I)) \tag{28}$$

And a maximum frequent pattern I is a closed pattern, that is not a subset of any other frequent patterns. According to this definition, the longest frequent patterns are also maximum frequent patterns. Since maximum frequent patterns is a subset of all the frequent patterns in a dataset, using methods that only mine maximum frequent pattern or closed frequent pattern will reduce pattern mining time.

7.2 Proof of Using Closed Frequent Patterns

Proof. The correctness of claim **C.3** can be proved: In each iteration of PatternKa algorithm 1, newly generated groups are removed from the original dataset, which reduces the dataset size. Since reducing the data size will only reduce pattern frequency in the dataset, therefore this process will not generate new frequent patterns:

$$\Pi_{before} = \Pi_{after} \tag{29}$$

Where Π_{before} is frequent pattern set before removing a group of samples from data, and Π_{after} is frequent pattern set after the removal.

Now we need to prove that data removal will not generate new closed frequent patterns. Assuming we have removed a subset of data G_r from dataset T and generates a new closed frequent pattern P_n:

$$T = T_{after} \bigcup G_r \tag{30}$$

Where T_{after} is the dataset after the removal of G_r. From equation 29 we have:

$$P_n \in \Pi_{before} \tag{31}$$

Since Π_{before} have closed frequent patterns Π_{closed} and non-closed frequent patterns $\Pi_{non-closed}$:

$$\Pi_{before} = \Pi_{closed} \bigcup \Pi_{non-closed} \tag{32}$$

It is trivial if $P_n \in \Pi_{closed}$, we only consider the case when

$$P_n \in \Pi_{non-closed} \tag{33}$$

Which means we only need to prove that a non-closed frequent pattern will not become closed frequent pattern after the removal of a subset of data.

We split data in G_r into 2 classes:

$$G_r = G_\phi \bigcup G_\epsilon \tag{34}$$

Where data in the first class G_ϕ does not contain pattern P_n, and data in the second class G_ϵ does:

$$G_\phi = \{t \mid t \in T, P_n \nsubseteq t\} \tag{35}$$

$$G_\epsilon = \{t \mid t \in T, P_n \subseteq t\} \tag{36}$$

The data in G_ϕ will not affect the closeness of P_n, we only need to consider the data in G_ϵ.

We firstly prove that removing G_ϵ will not convert non-closed pattern P_n into a closed frequent pattern. From equation 27 we have

$$g(P_n) = g_{after}(P_n) \bigcup G_\epsilon \tag{37}$$

Where $g(P_n)$ and $g_{after}(P_n)$ is the sample set containing P_n in T and T_{after} respectively. Since P_n is not a closed frequent pattern in T, in dataset T we have:

$$P_n \neq f(g(P_n)) \tag{38}$$

$$P_n \subset f(g(P_n)) \tag{39}$$

Since

$$\forall t \in g(P_n), f(g(P_n)) \subset t \tag{40}$$

Therefore

$$\forall t \in g_{after}(P_n), f(g(P_n)) \subset t \tag{41}$$

And

$$P_n \subset f(g(P_n)) \subseteq f(g_{after}(P_n)) \tag{42}$$

Then

$$P_n \neq f(g_{after}(P_n)) \tag{43}$$

Therefore P_n is not a closed frequent pattern after removing G_ϵ, this is contradictory to our assumption.

In conclusion, there will not be any new closed frequent pattern generated after the removal of G_r, therefore claim **C.3** is proved. □