Hiding Emerging Pattern with Local Recoding Generalization

Michael Wai-Kit Cheng

Abstract

Establishing strategic partnership often requires organizations to publish and share meaningful data to support collaborative business activities. An equally important concern for them is to protect sensitive patterns like unique emerging sales opportunities embedded in their data. In this paper, we contribute to the area of data sanitization by introducing an optimization-based local recoding methodology to hide emerging patterns from a dataset but with the underlying frequent itemsets preserved as far as possible. We propose a novel heuristic solution that captures the unique properties of hiding EPs *to carry out iterative local recoding generalization. Also, we propose a metric which measures (i) frequent-itemset distortion that quantifies the quality of published data and (ii) the degree of reduction in emerging patterns, to guide a bottom-up recoding process. We have implemented our proposed solution and experimentally verified its effectiveness with a benchmark dataset.*

Keywords: Emerging patterns, pattern hiding, data sanitization, frequent itemsets

1 Introduction

Organizations often publish and share their data to support business collaboration. In the context of marketing and sales, companies can leverage on the customer pools of each other for cross-selling so that the involved parties can gain sales volume increase. Due to the equally important need of privacy protection, customers often expect their data to be anonymized before sharing [26] and studies on privacy-preserving data publishing have bloomed [11]. Furthermore, trade secrets embedded in data are valuable to organizations [**?**] and needed to be properly protected. For instance, patterns like recent increase in the sales volume of a product line for a certain customer group (emerging marketing trends) can be an example. Leaking of related intelligence could cause company loss in gaining the first-mover advantage. Even though companies understand that data sharing is unavoidable to support collaborative activities like cross-selling, they may face a great hindrance to data sharing if the emerging sales opportunities of their own business cannot be hidden.

Among others, *emerging patterns* [18] embedded in data carry sensitive information, that data owners may prefer to hide. In fact, previous studies have revealed that emerging patterns are highly discriminative when used as features for classification [28, 10, 7], and thus carry salient features of the data. The hiding, however, is technically challenging as collaborative data analysis is still often expected to facilitate collaboration. That is, some statistical properties of the data to-be-shared are preserved as far as possible. In particular, frequent itemset mining has already been well-supported in most commercial data-mining packages. Therefore, in this paper, we study how to hide emerging patterns while preserving frequent itemsets.

To hide emerging patterns, we adopt recoding generalization methods. In particular, we adopt local recoding which is (intuitively) a value-grouping generalization process, given an attribute generalization hierarchy. To ensure that the generalized data neither (i) reveal sensitive information nor (ii) produce a highly distorted mining result, we propose metrics for quantifying the two competing objectives. With the metrics, we present an iterative, bottom-up optimization framework. Compared with hiding frequent itemsets [24], hiding emerging patterns is more technically challenging. In particular, the *a priori anti-monotone* property does not hold in emerging patterns. Thus, the search space of emerging patterns is huge. Worst still, a local recoding may hide an emerging pattern while generating new emerging patterns. To the best of our knowledge, there has not been work on hiding emerging patterns.

2 Related Work

Studies on data sanitization can be dated back to the earlier work on statistical disclosure control [1]. Recent development in privacy preserving data mining [25] has contributed to some advances in privacy measure and data sanitization method. For example, to avoid personal identity to be recovered from an anonymized demographic dataset, a number of privacy measures were proposed in the literature, e.g., k-anonymity [26] and ℓ-diversity [21]. Other

privacy measures include k^m-anonymity [**?**] and (h,k,p)-coherence [**?**]. Given a particular measure, recoding generalization [18, 13, 17, 25, 8, 30, 19] and perturbation [2, 9, 16] are two commonly adopted data sanitization approaches. Recoding generalization is often preferred over the perturbation approach as the dataset sanitized by recoding generalization is still semantically consistent with the original one, even though it is "blurred". While this study aims at hiding emerging patterns instead of personal identities, the concepts like equivalence classes and recoding generalization are adopted in the proposed methodology.

To control the distortion of the data caused by the sanitization, attempts have been made to preserve as much information of the original dataset as possible to, say, preserve the subsequent classification accuracy [14] and clustering structure [12]. In addition, there has been some recent work studying the tradeoff between privacy and utility [20] in the context of privacy-preserving data publishing. In this work, we try to preserve the frequent itemsets of the data as far as possible.

Recently, there has been work [24, 23] on hiding patterns like frequent itemsets where users specify a subset of frequent itemsets, namely sensitive frequent itemsets, that are not supposed to be disclosed to the public. In our study, we focus on hiding emerging patterns, which makes a unique contribution to the area of pattern hiding. Emerging patterns (EP) are features that are distinctive from one class to another and has been found to be effective for building highly accurate classifiers [15, 28, 10, 7]. Mining EPs from large databases is technically intriguing as the total number of EPs, in the worst case, is exponential to the total number of attributes in transactions, and there has not been a corresponding notion of the apriori anti-monotone property of frequent itemsets in EPs so that the search space can be pruned. Previous work on EPs mainly focuses only on the mining efficiency, e.g., using a border-based approach [4], a constraint-based approach [31], or focusing only on jumping EPs [3]. So far, there exists no related work on emerging pattern hiding.

3 Background and Problem Statement

In the following, we present the definitions, notations used and the problem statement.

A *transactional dataset* is a set of transactions. Let $I = \{i_1, i_2, ..., i_n\}$ be a finite set of distinct *items* in D. A *transaction* t has a set of nominal attributes $A = \{a_1, a_2, ..., a_m\}$ and each attribute a_i takes values from a set $V_i \subseteq I$. We make two simple remarks about these notations. (i) While we assume transactional data with nominal attributes, data of a continuous domain can be cast into nominal data, for example by defining ranges. (ii) One may consider a relation as a set of transactions of a fixed arity.

An *itemset* X is a (proper) subset of I. $Supp_D(X)$ denotes the support of an itemset X in a dataset D, which can be computed as $\frac{|\{t \mid X \in t \wedge t \in D\}|}{|D|}$. Given a support threshold σ, X is said to be a *σ-frequent itemset* if $Supp_D(X) \geq \sigma$. The growth rate of an itemset is the ratio of its support in one dataset to that in the other.

Definition 3.1: [6] Given two datasets, namely D_1 and D_2, the *growth rate* of an itemset X, denoted as $GR(X, D_1, D_2)$, from D_1 to D_2 is defined as $GR(X, D_1, D_2) =$

$$\begin{cases} 0 & \text{, if } Supp_{D_1} = 0 \text{ and } Supp_{D_2} = 0 \\ \infty & \text{, if } Supp_{D_1} = 0 \text{ and } Supp_{D_2} > 0 \\ \frac{Supp_{D_2}(X)}{Supp_{D_1}(X)} & \text{, otherwise.} \end{cases}$$

■

Definition 3.2: [6] Given a growth rate threshold ρ and two datasets D_1 and D_2, an itemset X is a *ρ-emerging pattern* (ρ-EP) from D_1 to D_2 if $GR(X, D_1, D_2) \geq \rho$. ■

Intuitively, given two datasets, emerging patterns (EPs) [6] are the itemsets whose support increases significantly from one dataset to another. The formal definition of EPs is presented in Defintion 3.2. An emerging pattern with a growth rate ∞ (i.e., itemset that appears in one dataset but not the other) is called a *jumping emerging pattern*.

For ease of presentation, we may skip σ, ρ, D_1 and D_2 of EPs when they are not essential to our discussions.

Example 3.1: Figure 1 (a) shows a simplified hypothetical dataset D of the Adult dataset [27]. It contains some census information of the United States. More description of the dataset can be found in Section 8. We opt to present some nominal attributes of Adult for discussions. Each record (or transaction) represents a person. Consider two subsets D_1 containing married people and D_2 containing those who do not. From Figure 1 (a), we find the following emerging patterns, among many others.

- The pattern (MSE, manager) has a support of 75% in D_1 and 20% in D_2. Therefore, the growth rate of (MSE, manager) from D_1 to D_2 is 3.75. When we set ρ to 3, (MSE, manager) is a ρ-emerging pattern in D_2.
- High-school graduate (HS) has a support of 0% in D_1 but 20% in D_2. Hence, its growth rate from D_2 to D_1 is infinite. (HS) is a jumping emerging pattern in D_1.

Next, we state the formal problem statement of this paper below (Figure 1 (b)). ■

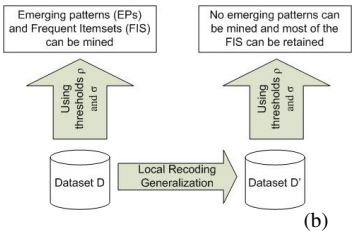
Problem statement. Given two datasets (D_1, D_2), σ and ρ, we want to sanitize (D_1, D_2) to (D_1', D_2') such that no ρ-EPs from D_1' to D_2' can be mined while the distortion between σ-frequent itemsets of (D_1, D_2) and those of (D_1', D_2') is minimized. ■

4 Multidimensional Local Recoding

Our algorithm for hiding emerging pattern is based on local recoding generalization (multi-local-recode in

ID	Edu.	Martial	Occup.	Rel.	Race	Sex
1	BA	married	executive	wife	black	F
2	MSE	married	manager	husband	black	M
3	MSE	married	manager	wife	white	F
4	MSE	married	manager	husband	black	M
5	BA	never	manager	NA	white	M
6	MSE	never	manager	NA	white	F
7	HS	never	repair	NA	black	M
8	BA	never	manager	NA	white	M
9	BA	never	manager	NA	black	F

(a)



(b)

Figure 1. (a) A hypothetical subset of `Adult` and (b) The problem statement illustration

Figure 2). In this section, we give an overview of recoding generalization, or *recoding* for simplicity.

Recoding. As discussed in Section 1, recoding has been proposed for anonymization. The idea of recoding is to modify values into more general ones such that more tuples will share the same values and cannot be distinguished individually. Thus, anonymization can be achieved. Here, we recode values in emerging patterns with some non-emerging values. Thus, the recoded patterns become less emerging.

Multidimensional local recoding. In this work, we adopt the notion of multidimensional local recoding [8, 30, 19], from the context of k-anonymity. It recodes values at "cell level". It relies on equivalence classes. An *equivalence class* of attributes A is a set of tuples T, where $\pi_A(T)$ is a singleton. That is, the tuples in T have the same value in attributes A. In a recoding, the tuples in an equivalence class (of a set of attributes) and those in another equivalence class are recoded into the lowest common ancestors along the hierarchies. One subtle point is that this recoding does not require the entire equivalence class to be recoded, as long as anonymity can be achieved. Hence, both original and generalized values may co-exist in the recoded dataset.

Example 4.1: Let us revisit the dataset in Figure 1 (a) and the emerging pattern (`MSE`, `manager`). The emerging pattern is related to the attributes of education background (`Edu.`) and occupation (`Occup.`). Regarding (`Edu.`, `Occup.`), the equivalence classes in D_2 are $\{\{5, 8, 9\}, \{6\}, \{7\}\}$, where the numbers are the IDs. In multidimensional lcoal recoding, we may recode the `Edu.` attribute of the subset of $\{2, 3, 4\}$ and $\{5, 8, 9\}$. For example, we may recode $\{2, 3, 4\}$ with $\{8, 9\}$. and we may recode `BA` and `MSE` into degree holder `Deg`. The growth rate of (`Deg.`, `manager`) in the recoded dataset is 75%/40% = 1.875. Hence, (`Deg.`, `manager`) is not ρ-emerging when ρ = 3. In addition, after such a recoding, all `BA`, `MSE` and `Deg.` appear in the recoded dataset. ∎

Other notions of recoding, including single-dimensional global recoding [5, 13, 17, 25] and multidimensional global recoding [18], generalize values in a relatively coarse granularity and very often result in over-generalization.

5 Algorithm for Hiding Emerging Patterns

In this section, we present the overall algorithm `hide-eps` (shown in Figure 2) for hiding emerging patterns with a minimal distortion in frequent itemsets.

Procedure `hide-eps`
Input: two datasets, D_i and D_j, the threshold of growth rate and frequent itemsets ρ and σ, the heuristic parameters p and q, an initial temperature t_0 and the cooling parameter α
Output: transformed datasets (D_i, D_j)

01 $t = t_0$ H.`init`() // initialization
02 E := `mine-eps`(D_i, D_j, ρ) // [31]
03 **while** $E \neq \emptyset$
04 F := `incr-mine-fis`$(D_i \cup D_j, \sigma)$ // [**?**]
05 e := `next-overlapping-ep`(E)
06 **if** e is not null
(D_i, D_j) := `local-recoding-sa`$(D_i, D_j, e, F, t, \alpha, H)$
07 **if** $t > 0.01$ **then** $t = \alpha \times t$
08 E := `mine-eps`(D_i, D_j, ρ)
09 **return** (D_i, D_j)

Procedure `local-recoding-sa`
Input: two datasets, D_i and D_j, an emerging pattern e, a frequent itemset F, a temperature t, a hashtable H for caching the utility gain of local recodings
Output: transformed datasets (D_i, D_j)

10 **let** D_i be the dataset where e has a higher support
11 denote c_e be the equiv. class of e in D_i
12 compute equiv. classes C of D_j of the attributes of e
// compute the utility gain of the local recoding of each equiv. class c_k in C with c_e
13 **for each** c_k in C
14 **if** `determine-missing-FIS`$(G_{(c_e,c_k)}, F) = \emptyset$ **then**
15 **if** `determine-new-singleton-eps`$(G_{(c_e,c_k)}, E) = \emptyset$ **then**
16 **if** $H[c_e][c_k]$ is null **then**
17 $H[c_e][c_k]$:= `util_gain`$(G_{(c_e,c_k)}, E)$ // Section 6
18 c_k := `get-next-step-sa`(c_e, H, t)
19 D_i := `multi-local-recode`(D_i, c_e, c_k) // Section 4
20 D_j := `multi-local-recode`(D_j, c_e, c_k)
21 **return** (D_i, D_j)

Figure 2. The overall algorithm

Overview of `hide-eps`. The main ideas of `hide-eps` can be described as follows. First, we determine the emerging patterns to be hidden (Line 02) and the frequent itemsets (incrementally) to be preserved (Line 04). We refer the details of Lines 02 and 04 to previous works [**?**, 31], since our focus is on *hiding* emerging patterns. For each selected emerging pattern (Line 05), we carry out a local recoding `local-recoding-sa` (Line 06, more details soon). This process (Lines 03-08) is repeated until there is no more emerging pattern to hide (Line 03). To avoid sub-optima, we present `hide-eps` in the style of simulated annealing search (Lines 01, 07 and 18).

Next, we discuss the details of the major steps of the algorithm.

Mining emerging patterns (`mine-eps`, Lines 02 and 08). During recoding, we invoke `mine-eps` [31] to determine if all the emerging patterns have been hidden (Line 08). To

the best of our knowledge, there does not exist any incremental algorithm for mining emerging patterns. As verified by our experiments, `mine-eps` is a bottleneck of runtime of `hide-eps`. However, it should be remarked that the emerging patterns may often be altered slightly by most local recodings, in practice. To address this performance issue, in Section 8, we tested another version of `hide-eps`, where `mine-eps` is invoked *only* when all previously mined emerging patterns have been hidden.

Incremental mining of frequent itemsets (`incr-mine-fis`, Line 04). A local recoding may alter the existing frequent itemsets. Figure 3 (b) (ii) shows an example. Since a local recoding changes only part of D_1 and D_2, we need not mine the dataset from scratch but do it incrementally using algorithms like [**?**].

Selecting emerging patterns for recoding (`next-overlapping-ep`, Line 05). Given a set of emerging patterns E, `next-overlapping-ep` determines the emerging pattern e in E such that it overlaps with the remaining emerging patterns the most. The intuition is that reducing the growth rate of e may indirectly reduce the growth rate of the overlapping emerging patterns as well. We verify with some experiments that this approach consistently outperforms a number of other strategies (see Section 8).

Determining the next local recoding (`local-recoding-sa`, Lines 06, 10-21). Assume that c_e is the equivalence class of the emerging pattern e. We first compute the equivalence classes of the attributes of e to generalize with c_k (Line 12). We apply the utility gain defined in Section 6 to determine the goodness of local recodings (Line 16). Since there can be many equivalence classes, this is another bottleneck of runtime. We speed up that step using (i) a hashtable (Lines 01 and 16-17) to cache the utility gain values computed, and (ii) two filters on equivalence classes (Lines 14 and 15). The first filter is that we ignore the equivalence classes that would result in missing frequent itemsets, which is obviously undesirable. This can be computed by the change in support of itemsets in F due to a local recoding. Second, we discard a local recoding that would yield new single-attribute emerging patterns. This can be computed by determining the growth rate of the equivalence classes with the attributes of e. We did not compute possible new multi-attribute emerging patterns because of its daunting complexity.

With the utility gain of equivalence classes, we use a simulated annealing search (`get-next-step-sa`, Line 18), as a black box, to get the next local recoding.

Analysis. Given that A_E is the set of attributes of the emerging pattern E, and $\mathcal{D}$ and $\mathcal{H}$ are the overall domain size and the maximum height of the hierarchy of all possible A_E's, respectively. In the worst case, there can be $O(\mathcal{D}\times\mathcal{H})$ possible recodings. Also, local recoding allows tuple-wise recoding and thus in the worst case, $|D_1|$ + $|D_2|$ recoding operations can be carried out. Thus, the search space of finding the optimal recoding is $O((|D_1|+|D_2|)\times\mathcal{D}\times\mathcal{H})$. This work proposes a heuristic search for this problem. While the loop (Lines 03-08) may repeat many times in the worst case, the number of iterations needed was found small in practice. As discussed, `mine-eps` and the computation of `util_gain` are the bottlenecks of runtime. The runtime of the former is experimentally evaluated in [31]. The time complexity for the latter is $O(|A_e|\times\mathcal{D}\times\mathcal{H}\times|F|)$, where $e\in E$, $|A_e|\times\mathcal{D}\times\mathcal{H}$ is the number of possible equivalence classes and for each class, $O(|E|)$ and $O(|F|)$ are used to compute RG_{local} and RD_{local}, respectively.

6 Metric for Multidimensional Local Recoding

In this section, we define an utility gain (`util_gain`) to quantify the effectiveness of a local recoding. `util_gain` will guide the process for hiding emerging patterns `local-recoding`, in Figure 2. A recoding is effective if (i) the distortion of frequent itemsets is small and (ii) the reduction in the growth rate of emerging patterns is large.

Metric for the distortion of frequent itemsets. For presentation clarity, we will present our proposed metric for global recoding followed by its adaption for local recoding.

(A) Distortion metric for single-dimensional global recoding. Single-dimensional global recoding performs recoding on the domain of an attribute in a dataset. It recodes a value of the domain to another (generalized) value. That is, if a particular value is recoded, the attribute of all the tuples containing that particular value will be recoded. No frequent itemsets disappear but may appear in a generalized form after a recoding (Figure 3 (a)).

Inspired by the distortion metric proposed in [19], we propose a metric for measuring the *recoding distance* ($RDist$) between the original and generalized form of a tuple. Then, we define a metric called *value distance* (VD) which measures the distance between the original and generalized form of a single attribute value. We will use VD as a building block for the definition of distortion (RD). Since a recoding always assumes an attribute hierarchy, we may skip the hierarchy H when it is clear from the context.

Definition 6.1: *Recoding Distance (RDist):* Consider a recoding G which generalizes a set of non-generalized values V to a single generalized value v_g, where V is the set of values under v_g in an attribute hierarchy. The recoding distance of G $RDist(G)$ is $|V|$. ∎

Definition 6.2: *Value Distance (VD):* Let h be the height of an attribute hierarchy H, where level h and 0 is the most generalized and specific level, respectively. Consider a value v at level p which is generalized to a value v' at level

q. Let G_i denotes the recoding that generalizes an attribute from level $i-1$ to i, where $0 < i \leq h$. The *value distance* between v and v' is: $VD(v, v') = \sum_{i=p}^{q} \frac{i \cdot RDist(G_i)}{h}$. ∎

Value distance is unfavorable to recoding (i) many values into one single generalized value; and (ii) a value into a generalized value that is close to the top of the hierarchy. This gives a measure for the distortion of a value due to a recoding. Next, we extend VD to measure the distortion of a tuple and frequent itemsets due to recoding.

Definition 6.3: *Tuple Distance (TD):* Suppose a tuple $f = (v_1, v_2, \ldots, v_n)$ is generalized to $f' = (v'_1, v'_2, \ldots, v'_n)$. The tuple distance between f and f' is defined as: $TD(f, f') = \sum_{i=1}^{n} VD(v_i, v'_i)$. ∎

Definition 6.4: *Recoding Distortion (RD):* Let $F = \{f_1, f_2 \ldots f_n\}$ be a set of *σ-frequent itemsets* in D and $F' = \{f'_1, f'_2 \ldots f'_m\}$ be the set of *σ-frequent itemsets* in D', where $m \leq n$. The corresponding frequent itemset of f_i due to global recoding is denoted as $f'_j = G(f_i)$. The recoding distance between F and F' is defined as: $RD(F, F') = \sum_{i=1}^{n} TD(f_i, G(f_i))$. ∎

Example 6.1: Following up Example 4.1, we compute the (global) recoding distortion of generalizing (MSE, manager) to (Deg., manager). Figure 3 shows the attribute hierarchy of Edu.The recoding distortion $RD(\{$(MSE, manager)$\}, \{$(Deg., manager)$\})$, RD, can be computed as follows: RD = $TD((\texttt{MSE}, \texttt{manager}), (\texttt{Deg.}, \texttt{manager}))$ = $VD(\texttt{MSE}, \texttt{Deg.}) + VD(\texttt{manager}, \texttt{manager})$ = $\sum_{i=0}^{2} \frac{i \cdot RDist(G_i)}{h} = \frac{1 \times 3}{2} + \frac{2 \times 0}{2}$ = 1.5 ∎

(B) Distortion metric for local recoding. Since single-dimensional global recoding may often lead to over-generalization, we adopted local recoding. We remark that there are two unique challenges in computing recoding distance for local recoding (Figure 3 (b)).

(B.i) An itemset in F having no correspondence in F'. Local recoding allows part of the tuples that share the same attribute values to be generalized. Such recoding may generalize some supporting tuples of a frequent itemset which makes the itemset (in the original or generalized form) not frequent anymore. To address this, we measure the distortion of the disappeared frequent itemset to the most general form. The reason is that the frequent itemset can be trivially recovered when the entire dataset is generalized to the most general form.

Specifically, given a f in F, if we cannot find a corresponding frequent itemset in F', we first create an itemset, f_{max}, which contains the most generalized value of each value in f. Then, RD of f is the recoding distance between f and f_{max}.

Example 6.2: Reconsider the dataset in Figure 1 (a). Suppose we recode the Edu. attribute of Records 1 and 2 to Deg. When σ is 40%, {BA} and {MSE} were frequent itemsets (not minimal for illustration purposes) before recoding and there is no frequent itemset after recoding. ∎

(B.ii) An itemset in F having more than one corresponding itemset in F'. As discussed, local recoding may generalize a frequent itemset f in F into more than one correspondence in F', denoted as F_f. In this case, we calculate the tuple distance of each of the corresponding itemsets in F_f and take the *minimum* tuple distance as the tuple distance of f. This is because the itemset with the minimum tuple distortion has been revealed in F', even when there may be more distorted itemsets.

With the above, we have the following recoding distance for local recoding:

Definition 6.5: *Recoding Distance for Local Recoding (RD_{local}):* Let $F = \{f_1, f_2 \ldots f_n\}$ be a set of *σ-frequent itemsets* in D and $F' = \{f'_1, f'_2 \ldots f'_m\}$ be the set of *σ-frequent itemsets* in D'. The corresponding frequent itemset(s) of f_i due to local recoding is denoted as $F_f = G(f_i)$. The recoding distance between F and F' is: $RD_{local}(F, F') = \frac{1}{n} \sum_{i=1}^{n} \frac{TD_{local}(f_i, G(f_i))}{TD_{local}(f_i, f_{max})}$, where $TD_{local}(f_i, G(f_i))$ =

$$\begin{cases} \theta_q \times TD(f_i, G(f_i)), & \text{if } f \text{ has 1 correspondent in } F' \\ (1-\theta_q) \times TD(f_i, f_{max}), & \text{if } f \text{ has no correspondent in } F' \\ \theta_q \times min(TD(f_i, f_j)), & \\ \qquad where\ f_j \in G(f_i), & \text{otherwise,} \end{cases}$$

θ_q is a parameter that specifies the relative importance of the itemset distortion and missing itemsets, due to G, and $TD_{local}(f_i, f_{max})$ is for normalizing RD_{local}. ∎

Example 6.3: Following up Example 6.2, when σ is 30%, the frequent itemset {(BA)} corresponds to the frequent itemsets {(BA), (Deg)} in the recoded datasets. ∎

Metric for the change in growth rate. The second component of our heuristics concerns the growth rate of the emerging patterns. Intuitively, we aim at a recoding that significantly reduces the growth rate of the emerging patterns in order to hide them. Given an emerging pattern e and the result of a local recoding e', the reduction in growth rate due to the recoding can be easily defined as the growth rate of e minus the growth rate of e'. Then, the growth rate reduction of E due to a local recoding G, denoted as $RG_{local}(G, E)$, can be defined as the total reduction in the growth rate of e in E divided by the total growth rate of e in E.

Putting all these together. Based on the derivations above, the utility gain due to a local recoding G for a set of emerging patterns E is defined as:
$\texttt{util\_gain}(G, E) = \theta_p RG_{local}(G, E) - (1-\theta_p) RD_{local}(F, F')$.
The two parameters θ_p and θ_q, where θ_p, $\theta_q \in$ [0,1], are specified by users.

7 Implementation Optimization

In this section, we discuss some implementation issues for optimizing the computation of the proposed algorithm.

Figure 3. (a) An attribute hierarchy of `Edu.`; and (b) the relationship between F and F' in (i) global recoding and (ii) local recoding

1. The maintenance of the equivalence classes. Local recoding relies on equivalence classes (Lines 03, 06 and 07 in `local-recoding` in Figure **??**). A local recoding would change the equivalence classes in the datasets slightly. We used a hashtable to keep track of the update of the classes caused by a recoding (Line 07). Since an emerging pattern may not be hidden by simply one recoding, we note that we may compute the equivalence class of attributes multiple times (Line 03). With the hashtable, Line 03 does not recompute existing equivalence classes.

2. An index for checking correspondents of F in F'. Given two sets of frequent itemsets F and F', we check the correspondences of an itemset in F to measure distortion between F and F' (`util_gain`). The core of this is to check whether a value v is a generalized version of another value in an attribute hierarchy. While an attribute hierarchy is often a small tree, these checks occur in every iteration of `hide-eps`. We apply an index [29] for computing the ancestor-descendant relationship of nodes in a tree. This significantly reduces the runtime of our algorithm.

3. A data structure for checking correspondents of E in E'. The other task in computing `util_gain` is to keep track of the change of emerging patterns during local recoding, which is necessary to measure the reduction in growth rate. Hence, we associate an emerging pattern of e to its records. By comparing the records of e and e', we obtain the correspondence between e and e'.

8 Experimental Evaluation

To verify the effectiveness and efficiency of our proposed algorithms, we conducted several experiments on `Adult` dataset [27] using the attribute hierarchies from [13].

We implemented our algorithm in `JAVA SDK` 1.6[1]. We have run our experiments on a PC with a Quad `CPU` at 2.4`GHz` and 3.25`GB RAM`. The PC is running Windows XP operating system. We have used system calls to invoke the implementations from [31] and [22] to determine emerging patterns and frequent itemsets, respectively.

The simplified `Adult` dataset contains 8 attributes. We removed the records with missing values. The records in the dataset were divided into two classes - people who have more than \$50k/year (7508 records) and people who do not (22654 records).

The effect of the parameters θ_p and θ_q. The first experiment is to verify the effects on the parameters θ_p and θ_q on the heuristic algorithm. In this experiment, we do not apply any filter (i.e., Lines 14-15 in `hide-eps`) and `SA` search (Line 18) in order to observe the effects on the parameters clearly. Instead, we used a `Greedy` search. The *performance* was presented in *"distortion on the frequent itemsets / the number of missing frequent itemsets", unless otherwise specified*. When σ and ρ were set to 40% and 5, respectively, the frequent itemsets obtained are: {(`Husband`, `Married-civ-spouse`, `Male`), (`Married-civ-spouse`, `White`), (`Married-civ-spouse`, `United-States`), (`Male`, `Private`, `White`), (`Male`, `Private`, `United-States`), (`Male`, `White`, `United-States`), (`Private`, `White`, `United-States`)}.

When we recode all attributes to `All` in the frequent itemsets, we obtain the maximum distortion of the frequent itemsets of `Adult` 623.1.

To illustrate the possible effect of recodings on the resulting frequent itemsets, we list the frequent itemsets after we applied `Greedy`, where θ_p and θ_q were both set to 0.8: {(`Relationship`, `United-States`), (`Married`, `White`, `United-States`), (`Male`, `Private`, `White`), (`Male`, `Private`, `United-States`), (`Male`, `White`, `United-States`), `Private`, `White`, `United-States`)}. The distortion obtained is 21.5 (out of 623.1).

Next, we varied θ_p and θ_q and measured the performance of `Greedy`. The performance is shown in Table 1 (LHS). The average runtime is 58 mins and 16 out of 58 mins is spent on mining `EPs`. The average number of local recodings is 14.

We make four observations from Table 1 (LHS). Firstly, when θ_p is set to 0, the algorithm concerns only distortion (regardless the corresponding reduction in growth rate) during local recodings. In such a case, the distortion on frequent itemsets of various θ_q's is in general large. The reason is that when θ_p is 0, the heuristics does not effectively reduce the growth rate and the search takes more recodings that are not relevant to hiding emerging patterns. Secondly, when θ_p is 1, the algorithm concerns only the reduction in growth rate. Note that 3 out of 7 frequent itemsets have disappeared. Thirdly, when we set θ_q to 1, we do not concern the missing frequent itemsets. Hence, more frequent item-

[1]The implementation is available at http://www.comp.hkbu.edu.hk/∼michael/source.rar.

Table 1. The effect of the parameters in `util_gain` on `Greedy`'s performance (LHS) and the performance of the `determine-new-singleton-eps` filter (RHS)

$\theta_p \backslash \theta_q$	0.0	0.2	0.4	0.6	0.8	1.0
0	73.3/1	50.0/1	50.0/1	50.0/1	50.0/3	50.0/1
0.2	73.3/1	59.7/1	38.2/1	38.2/1	46.5/1	11.5/4
0.4	73.3/1	59.7/1	38.2/1	21.5/1	46.5/1	11.5/4
0.6	73.3/1	59.7/1	21.5/1	38.2/1	46.5/1	11.5/4
0.8	73.3/1	59.7/1	21.5/1	21.5/1	38.2/1	0/5
1.0	11.5/3	11.5/3	11.5/3	11.5/3	11.5/3	11.5/3
$\theta_p \backslash \theta_q$	0.0	0.2	0.4	0.6	0.8	1.0
0	62.7/0	43.7/1	43.7/1	35.8/3	35.8/3	11.1/4
0.2	62.7/0	32.0/1	32.0/1	15.7/2	9.7/3	7.5/4
0.4	62.7/0	32.0/1	16.8/1	21.9/2	9.7/3	7.5/4
0.6	71.5/0	32.0/1	16.8/1	21.9/2	7.2/3	0/5
0.8	71.5/0	32.0/1	16.8/1	15.7/2	7.2/3	0/5
1.0	73.1/1	73.1/1	73.1/1	73.1/1	73.1/1	73.1/1

sets were lost. Similarly, when we set θ_q to 0, we concern only the frequent itemsets that do not disappear. Since there is one missing frequent itemset during recodings, overlooking this led to more distortion. Fourthly, we found that a significant runitme (42 mins) was spent on calculating the utility gain of equivalence classes. The reason is that no filters had been applied yet.

In all, we found that on `Adult`, `Greedy` yields frequent itemsets with a distortion 21.5 (out of 623.1) and 1 missing frequent itemset when both θ_p and θ_q are moderate.

The effect of the `determine-new-singleton-eps` filter. This filter is used to avoid recoding equivalence classes that would yield new single-attribute `EP`s (Line 15 of `hide-eps`). The performance of `Greedy` with this filter is shown in Table 1 (RHS).

We observe from Table 1 (RHS) that there are similar trends on the performance with various θ_p and θ_q. The distortion is sometimes smaller but the missing frequent itemsets may sometimes be more. However, the average runtime of this experiment is 26 mins (compared to 58 previously). Specifically, the time for computing utility gain has been reduced from 42 to 15 mins. The number of recodings reduces from 14 to 9. The runtime improvement is due to (i) the smaller number of equivalence classes for computing the utility gain and (ii) fewer (if any) new `EP`s generated during `hide-eps`.

The effect of the `determine-missing-FIS` filter. From the previous experiments, we note that there are missing frequent itemsets in most cases. Here, we test the effectiveness of `determine-missing-FIS` filter (Line 14). The performance is shown in Table 2 (LHS). The average runtime for computing the utility gain increased from 15 to 21 mins and the number of recodings increased from 9 to 14. At first glance, the distortion might have increased. However, there is no missing frequent itemset for all θ_p's and θ_q's. This improvement comes at the expense of a slight increase in runtime.

The effect of calling `mine-eps` when E is empty. In the last experiment, 15 out of 36 mins was spent on mining `EP`s. In this experiment, we attempt to improve the runtime by hiding all existing `EP`s first before calling `mine-eps`, as opposed to calling `mine-eps` after each recoding. The performance is shown in Table 2 (RHS). From the result, we found that there is a slight increase in distortion. However, the time for mining `EP`s is reduced from 15 to 8 mins. The average runtime for computing the utility gain increased from 21 to 23 mins and the number of iterations remains unchanged.

Table 2. The performance of the `determine-missing-FIS` filter (LHS) and the performance of invoking `mine-eps` only when E is empty (RHS)

$\theta_p \backslash \theta_q$	0.0	0.2	0.4	0.6	0.8	1.0
0	NA	89.2/0	89.2/0	89.2/0	71.5/0	71.5/0
0.2	105.3/0	89.2/0	78.6/0	78.6/0	41.5/0	41.5/0
0.4	105.3/0	78.6/0	50/0	50/0	41.5/0	41.5/0
0.6	105.3/0	78.6/0	50/0	50/0	41.5/0	41.5/0
0.8	105.3/0	78.6/0	61.3/0	61.3/0	41.5/0	41.5/0
1.0	105.3/0	105.3/0	105.3/0	105.3/0	105.3/0	105.3/0
$\theta_p \backslash \theta_q$	0.0	0.2	0.4	0.6	0.8	1.0
0	NA	97.2/0	97.2/0	81.3/0	70.1/0	81.3/0
0.2	105.3/0	97.2/0	97.6/0	81.3/0	59.3/0	59.3/0
0.4	105.3/0	97.6/0	64.9/0	64.9/0	59.3/0	59.3/0
0.6	105.3/0	78.6/0	64.9/0	64.9/0	59.3/0	59.3/0
0.8	105.3/0	78.6/0	64.9/0	70.1/0	59.3/0	59.3/0
1.0	105.3/0	105.3/0	105.3/0	105.3/0	105.3/0	105.3/0

Table 3. The performance of hiding the `EP` with the minimum overlapping (LHS) and the performance of simulated annealing search (RHS)

$\theta_p \backslash \theta_q$	0.0	0.2	0.4	0.6	0.8	1.0
0	NA	101.8/0	101.8/0	95.6/0	95.6/0	95.6/0
0.2	127.1/0	98.3/0	75.8/0	75.8/0	53.7/0	53.7/0
0.4	127.1/0	98.3/0	50.0/0	50.0/0	53.7/0	53.7/0
0.6	127.1/0	78.6/0	67.2/0	58.5/0	53.7/0	53.7/0
0.8	127.1/0	80.2/0	61.3/0	65.3/0	48.1/0	48.1/0
1.0	127.1/0	127.1/0	127.1/0	127.1/0	127.1/0	127.1/0
$\theta_p \backslash \theta_q$	0.0	0.2	0.4	0.6	0.8	1.0
0	67.4/0	27.8/0	58.5/0	50.0/0	44.3/0	23.6/0
0.2	80.1/0	62.5/0	22.4/0	53.8/0	47.8/0	52.1/0
0.4	84.1/0	81.4/0	55.7/0	29.3/0	53.7/0	37.4/0
0.6	85.0/0	50.0/0	97.0/0	31.8/0	40.5/0	22.8/0
0.8	64.9/0	45.2/0	30.0/0	59.6/0	39.5/0	32.3/0
1.0	84.1/0	31.7/0	47.9/0	44.1/0	28.9/0	51.6/0

The effect of hiding the EP with the minimum overlapping. To justify the decision of hiding the maximum overlapping `EP` in Line 05 of `hide-eps`, we conducted an experiment which first hides the `EP` with the minimum overlapping. The result is shown in Table 3 (LHS). We observed that the distortion is slightly larger than that of the maximum overlapping. However, the average runtime for computing the utility gain increased from 23 to 39 mins and the time for mining `EP` increased from 8 to 23 mins.

Simulated annealing search. After demonstrating the effects of various settings with `Greedy`, we applied `SA` on the algorithm (Line 18 of `hide-eps`). We set a low temperature (T=10) of `SA` with a high cooling rate (α=0.4). Hence, `SA` initially has some chances to avoid local sub-optima and

then converges to `Greedy` quickly. To explore the search space more, each `SA` was allowed to restart fifty times. The results are shown in Table 3 (RHS). `SA` introduces some randomness in the performance. Compared to the best versions (Table 2), `SA` often produces better results, at the expense of runtime.

9 Conclusions

We presented a heuristic local-recoding algorithm for hiding emerging patterns of a dataset while preserving its frequent itemsets as far as possible. We tested our algorithm with a benchmark dataset and showed its effectiveness.

References

[1] N. R. Adam and J. C. Worthmann. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989.

[2] D. Agrawal and C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proc. of PODS*, 2001.

[3] J. Bailey, T. Manoukian, and K. Ramamohanarao. Fast algorithms for mining emerging patterns. In *Proc. of ECML/PKDD*, 2002.

[4] J. R. Bayardo. Efficiently mining long patterns from databases. In *Proc. of SIGMOD*, pages 85–93, 1998.

[5] R. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *Proc. of ICDE*, pages 217–228, 2005.

[6] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proc. of SIGKDD*, pages 43–52, 1999.

[7] G. Dong, X. Zhang, and L. Wong. CAEP: Classification by aggregating emerging patterns. In *Proc. of DS'99*, pages 30–42, 1999.

[8] Y. Du, T. Xia, Y. Tao, D. Zhang, and F. Zhu. On multidimensional k-anonymity with local recoding generalization. In *Proc. of ICDE*, pages 1422–1424, 2007.

[9] A. Evfimievski, R. Strikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proc. of SIGKDD*, 2002.

[10] H. Fan and K. Ramamohanarao. A Bayesian approach to use emerging patterns for classification. In *Proc. of ADC*, pages 39–48, 2003.

[11] B. Fung, K. Wang, A. Fu, and P. Yu. *Privacy-Preserving Data Publishing: Concepts and Techniques*. Chapman & Hall/CRC, 2010.

[12] B. Fung, K. Wang, L. Wang, and M. Debbabi. A framework for privacy-preserving cluster analysis. In *Proc. of ISI*, page 4651, 2008.

[13] B. Fung, K. Wang, and P. Yu. Top-down specialization for information and privacy preservation. In *Proc. of ICDE*, pages 205–216, 2005.

[14] B. Fung, K. Wang, and P. Yu. Anonymizing classification data for privacy preservation. *TKDE*, 10(5):711–725, 2007.

[15] H. F. K. Ramamohanarao. Pattern based classifiers. In *Proc. of WWW*, pages 71–83, 2007.

[16] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. Random-data perturbation techniques and privacy-preserving data mining. *KAIS*, 7(4):387–414, 2005.

[17] K. LeFevre, D. Dewitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *Proc. of SIGMOD*, pages 49–60, 2005.

[18] K. LeFevre, D. Dewitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *Proc. of ICDE*, page 25, 2006.

[19] J. Li, R. Wong, A. Fu, and J. Pei. Anonymization by local recoding in data with attribute hierarchical taxonomies. *TKDE*, 20(9):1181–1194, 2008.

[20] T. Li and N. Li. On the tradeoff between privacy and utility in data publishing. In *Proc. of SIGKDD*, 2009.

[21] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Vĕnkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *TKDD*, 1(1):3, 2007.

[22] MAFIA. *Mining Maximal Frequent Itemsets*. http://himalaya-tools.sourceforge.net/Mafia/.

[23] G. Moustakides and V. Verykios. A maxmin approach for hiding frequent itemsets. *DKE*, 65(1):75–79, 2008.

[24] S. Oliveira and O.R.Zaiane. Privacy preserving frequent itemset mining. In *Proc. of ICDM Workshop on Privacy, Security and Data Mining*, volume 14, pages 43–54, 2002.

[25] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *IJUFKS*, 10(5):571–588, 2002.

[26] L. Sweeney. k-anonymity: A model for protecting privacy. In *IJUFKS*, pages 557–570, 2002.

[27] UCI Machine Learning Repository. *Adult Data Set*. http://archive.ics.uci.edu/ml/datasets/Adult.

[28] Z. Wang, H. Fan, and K. Ramamohanarao. Exploiting maximal emerging patterns for classification. In *Proc. of AUS-AI*, pages 1062–1068, 2004.

[29] X. Wu, M. L. Lee, and W. Hsu. A prime number labeling scheme for dynamic ordered xml trees. In *Proc. of ICDE*, pages 66–78, 2004.

[30] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A. Fu. Utility-based anonymization using local recoding. In *Proc. of SIGKDD*, pages 785–790, 2006.

[31] X. Zhang, G. Dong, and K. Ramamohanarao. Exploring constraints to efficiently mine emerging patterns from large high-dimensional datasets. In *Proc. of SIGKDD*, pages 310–314, 2000.