Noname manuscript No.
(will be inserted by the editor)

Local and Global Recoding Methods for Anonymizing Set-valued Data

Manolis Terrovitis · Nikos Mamoulis · Panos Kalnis

Received: date / Accepted: date

Abstract In this paper we study the problem of protecting privacy in the publication of set-valued data. Consider a collection of supermarket transactions that contains detailed information about items bought together by individuals. Even after removing all personal characteristics of the buyer, which can serve as links to his identity, the publication of such data is still subject to privacy attacks from adversaries who have partial knowledge about the set. Unlike most previous works, we do not distinguish data as sensitive and non-sensitive, but we consider them both as potential quasi-identifiers and potential sensitive data, depending on the point of view of the adversary. We define a new version of the k-anonymity guarantee, the k^m-anonymity, to limit the effects of the data dimensionality and we propose efficient algorithms to transform the database. Our anonymization model relies on generalization instead of suppression, which is the most common practice in related works on such data. We develop an algorithm which finds the optimal solution, however, at a high cost which makes it inapplicable for large, realistic problems. Then, we propose a greedy heuristic, which performs generalizations in an Apriori, level-wise fashion. The heuristic scales much better and in most of the cases finds a solution close to the optimal. Finally, we investigate the application of techniques that partition the database and perform anonymization locally, aiming at the reduction of the memory consumption and further scalability. A thorough experimental evaluation with real datasets shows that a vertical partitioning approach achieves excellent results in practice.

Manolis Terrovitis
IMIS
Research Center "Athena"
E-mail: mter@imis.athena-innovation.gr

Nikos Mamoulis
Dept. of Computer Science
University of Hong Kong
E-mail: nikos@cs.hku.hk

Panos Kalnis
Div. of Mathematical and Computer Sciences and Engineering
King Abdullah University of Science and Technology
E-mail: panos.kalnis@kaust.edu.sa

1 Introduction

We consider the problem of publishing set-valued data, while preserving the privacy of individuals associated to them. Consider a database D, which stores information about items purchased at a supermarket by various customers. We observe that the direct publication of D may result in unveiling the identity of the person associated with a particular transaction, if the adversary has some partial knowledge about a subset of items purchased by that person. For example, assume that Bob went to the supermarket on a particular day and purchased a set of items including coffee, bread, brie cheese, diapers, milk, tea, scissors, light bulb. Assume that some of the items purchased by Bob were on top of his shopping bag (e.g., brie cheese, scissors, light bulb) and were spotted by his neighbor Jim, while both persons were on the same bus. Bob would not like Jim to find out other items that he bought. However, if the supermarket decides to publish its transactions and there is only one transaction containing brie cheese, scissors, and light bulb, Jim can immediately infer that this transaction corresponds to Bob and he can find out his complete shopping bag contents.

This motivating example stresses the need to transform the original transactional database D to a database D' before publication, in order to avoid the association of specific transactions to a particular person or event. In practice, we expect the adversary to have only partial knowledge about the transactions (otherwise, there would be little sensitive information to hide). On the other hand, since the knowledge of the adversary is not known by the data publisher, it makes sense to define a generic model for privacy, which guarantees against adversaries having knowledge limited to a level, expressed as a parameter of the model.

In this paper, we propose such a k^m-anonymization model, for transactional databases. Assuming that the maximum knowledge of an adversary is at most m items in a specific transaction, we want to prevent him from distinguishing the transaction from a set of k published transactions in the database. Equivalently, for any set of m or less items, there should be at least k transactions, which contain this set, in the published database D'. In our example, Jim would not be able to identify Bob's transaction in a set of 5 transactions of D', if D' is 5^3-anonymous.

This anonymization problem is quite different compared to well-studied privacy preservation problems in the literature. Unlike the k-anonymity problem in relational databases [24, 25], there is no fixed, well-defined set of quasi-identifier attributes and sensitive data. A subset of items in a transaction could play the role of the quasi-identifier for the remaining (sensitive) ones and vice-versa. Another fundamental difference is that transactions have variable length and high dimensionality, as opposed to a fixed set of relatively few attributes in relational tuples. Finally, we can consider that all items that participate in transactions take values from the same domain (i.e., complete universe of items), unlike relational data, where different attributes of a tuple have different domains.

The m parameter in the k^m guaranty introduces a degree of flexibility in the traditional k-anonymity. It allows considering different scenarios of privacy threats and adjusting the guaranty depending on the severity of the threat and the sensitivity of the data. k^m-anonymity simulates k-anonymity if m is set equal to the domain size, i.e., the adversary knows all items. Still, this is not a realistic scenario in the transactional context. Since all items can act as quasi-identifiers, an attacker who knows them all and can link them to a specific person has nothing to learn from the original database. Her background knowledge already contains the original data.

To solve the k^m-anonymization problem for a transactional database, we follow a generalization approach. We consider a domain hierarchy for the items that participate in transactions. If the original database D does not meet the k^m-anonymity requirement, we gradually transform D, by replacing precise item descriptions with more generalized ones. For example, "skimmed milk" could be generalized to "milk" and then to "dairy product" if necessary. By carefully browsing into the lattice of possible item generalizations, we aim at finding a set of item generalizations, which satisfies the k^m-anonymity requirement, while retaining as much detail as possible to the published data D'.

We propose three classes of algorithms in this direction. Our first algorithmic class is represented by the *optimal anonymization* (OA) algorithm, which explores in a bottom-up fashion the lattice of all possible combinations of item generalizations, and finds the most detailed such sets of combinations that satisfy k^m-anonymity. The best combination is then picked, according to an information loss metric. Although optimal, OA has very high computational cost and cannot be applied for realistic databases with thousands of items. Motivated by this observation, we propose a second class of heuristic algorithms, which greedily identify itemsets that violate the anonymity requirement and choose generalization rules that fix the corresponding problems.

The first *direct anonymization* (DA) heuristic operates directly on m-sized itemsets found to violate k-anonymity. Our second, *apriori anonymization* (AA) is a more carefully designed heuristic, which explores the space of itemsets, in an Apriori, bottom-up fashion. AA first identifies and solves anonymity violations by $(l-1)$-itemsets, before checking l-itemsets, for l=2 to m. By operating in such a bottom-up fashion, the combinations of itemsets that have to be checked at a higher level can be greatly reduced, as in the meanwhile detailed items (e.g., "skimmed milk", "choco-milk", "full-fat milk") could have been generalized to more generalized ones (e.g., "milk"), thus reducing the number of items to be combined. Our experimental evaluation, using real datasets, shows that AA is a practical algorithm, as it scales well with the number of items and transactions, and finds a solution close to the optimal in most tested cases.

AA does not operate under limited memory, so there is still a case where the database will be large enough to prohibit its execution. We introduce a third class of algorithms, Local Recoding Anonymization (LRA) and Vertical Partitioning Anonymization (VPA), which anonymize the dataset in the presence of limited memory. The basic idea in both cases is that the database is partitioned and each part is processed separately from the others. A short version of this paper covering the first two classes of algorithms appears in [26].

The rest of the paper is organized as follows. Section 2 describes related work and positions this paper against it. Section 3 formally describes the problem that we study, provides an analysis for its solution space, and defines the information loss metric we use. In Section 4, we describe the algorithms and the data structures used by them. In Section 5 we discuss the advanced concepts of negative knowledge and ℓ^m-diversity. Section 6 includes an experimental evaluation, and Section 7 concludes the paper.

2 Related Work

Anonymity for relational data has received considerable attention due to the need of several organizations to publish data (often called microdata) without revealing the identity of individual records. Even if the identifying attributes (e.g., name) are removed, an attacker may be able to associate records with specific persons using combinations of other attributes (e.g., $\langle zip, sex, birthdate \rangle$), called quasi-identifiers (QI). A table is k-anonymized if each record is indistinguishable from at least $k-1$ other records with respect to the QI set [24,25]. Records with identical QI values form an anonymized group. Two techniques to preserve privacy are generalization and suppression [25]. Generalization replaces their actual QI values with more general ones (e.g., replaces the city with the state); typically, there is a generalization hierarchy (e.g., city$\rightarrow$state$\rightarrow$country). Suppression excludes some QI attributes or entire records (known as outliers) from the microdata.

The privacy preserving transformation of the microdata is referred to as *recoding*. Two models exist: in *global recoding*, a particular detailed value must be mapped to the same generalized value in all records. *Local recoding*, on the other hand, allows the same detailed value to be mapped to different generalized values in each anonymized group. The recoding process can also be classified into *single-dimensional*, where the mapping is performed for each attribute individually, and *multi-dimensional*, which maps the Cartesian product of multiple attributes. Our work is based on global recoding and can be roughly considered as single-dimensional (although this is not entirely accurate), since in our problem all items take values from the same domain.

[18] proved that optimal k-anonymity for multidimensional QI is NP-hard, under both the generalization and suppression models. For the latter, they proposed an approximate algorithm that minimizes the number of suppressed values; the approximation bound is $O(k \cdot logk)$. [3] improved this bound to $O(k)$, while [22] further reduced it to $O(\log k)$. Several approaches limit the search space by considering only global recoding. [5] proposed an optimal algorithm for single-dimensional global recoding with respect to the Classification Metric (CM) and Discernibility Metric (DM), which we discuss in Section 3.3. *Incognito* [14] takes a dynamic programming approach and finds an optimal solution for any metric by considering all possible generalizations, but only for global, full-domain recoding. Full-domain means that all values in a dimension must be mapped to the same level of hierarchy. For example, in the country$\rightarrow$continent$\rightarrow$world hierarchy, if Italy is mapped to Europe, then Thailand must be mapped to Asia, even if the generalization of Thailand is not necessary to guarantee anonymity. A different approach is taken in [21], where the authors propose to use *natural* domain generalization hierarchies (as opposed to user-defined ones) to reduce information loss. Our optimal algorithm is inspired by Incognito; however, we do not perform full-domain recoding, because, given that we have only one domain, this would lead to unacceptable information loss due to unnecessary generalization. As we discuss in the next section, our solution space is essentially different due to the avoidance of full-domain recoding. The computational cost of Incognito (and that of our optimal algorithm) grows exponentially, so it cannot be used for more than 20 dimensions. In our problem, every item can be considered as a dimension. Typically, we have thousands of items, therefore we develop fast greedy heuristics (based on the same generalization model), which are scalable to the number of items in the set domain.

Several methods employ multidimensional local recoding, which achieves lower information loss. *Mondrian* [15] partitions the space recursively across the dimension with the widest normalized range of values and supports a limited version of local recoding. [2] model the problem as clustering and propose a constant factor approximation of the optimal solution, but the bound only holds for the Euclidean distance metric. [29] propose agglomerative and divisive recursive clustering algorithms, which attempt to minimize the NCP metric (to be described in Section 3.3). Our problem is not suitable for multidimensional recoding (after modeling sets as binary vectors), because the dimensionality of our data is too high; any multidimensional grouping is likely to cause high information loss due to the dimensionality curse. [20,19] studied multirelational k-anonymity, which can be translated to a problem similar to the one studied here, but still there is the fundamental separation between sensitive values and quasi-identifiers. Moreover there is the underlying assumption that the dimensionality of the quasi-identifier is limited, since the authors accept the traditional unconditional definition of k-anonymity.

In general, k-anonymity assumes that the set of QI attributes is known. Our problem is different, since any combination of m items (which correspond to attributes) can be used by the attacker as a quasi-identifier. Recently, the concept of ℓ-diversity [17] was introduced to address the limitations of k-anonymity. The latter may disclose sensitive information when there are many identical sensitive attribute (SA) values within an anonymized group (e.g., all persons suffer from the same disease). [28, 31, 7] present various methods to solve the ℓ-diversity problem efficiently. [8] extends [28] for transactional datasets with a large number of items per transaction, however, as opposed to our work, distinguishes between non-sensitive and sensitive attributes. This distinction allows for a simpler solution that the one required in out setting, since the QI remains unchanged for all attackers. [30] also considers transactional data and distinguishes between sensitive and non-sensitive items, but assumes that the attacker has limited knowledge of up to p non-sensitive items in each transaction (i.e., p similar to our parameter m). Given this, [30] aims at satisfaction of ℓ-diversity, when anonymizing the dataset and requires that the original support of any itemset retained in the anonymized database is preserved. A greedy anonymization technique is proposed, which is based on suppression of items that cause privacy leaks. Thus, the main differences of our work to [30] is that we do not distinguish between sensitive and public items and that we consider generalization instead of suppression, which in general results in lower information loss. [16] proposes an extension of ℓ-diversity, called *t-closeness*. Observe that in ℓ-diversity the QI values of all tuples in a group must be the same, whereas the SA values must be different. Therefore, introducing the ℓ-diversity concept in our problem is a challenging, if not infeasible, task, since any attribute can be considered as QI or SA, leading to contradicting requirements. In Section 5 we discuss this issue in more detail.

Related issues were also studied in the context of data mining. [27] consider a dataset D of transactions, each of which contains a number of items. Let S be a set of association rules that can be generated by the dataset, and $S' \subset S$ be a set of association rules that should be hidden. The original transactions are altered by adding or removing items, in such a way that S' cannot be generated. This method requires the knowledge of S' and depends on the mining algorithm, therefore it is not applicable to our problem. Another approach is presented in [4], where the data owner generates a set of anonymized association rules, and publishes them instead of the original dataset. Assume a rule $a_1a_2a_3 \Rightarrow a_4$ (a_i's are items) with support $80 \gg k$ and confidence 98.7%. By definition, we can calculate the support of itemset $a_1a_2a_3$ as $80/0.987 \simeq 81$, therefore we infer that $a_1a_2a_3\neg a_4$ appears in $81 - 80 = 1$ transaction. If that itemset can be used as QI, the privacy of the corresponding individual is compromised. [4] presents a method to solve the inference problem which is based on the apriori principle, similar to our approach. Observe that inference is possible because of publishing the rules instead of the transactions. In our case we publish the anonymized transactions, therefore the support of $a_1a_2a_3\neg a_4$ is by default known to the attacker and does not constitute a privacy breach.

Concurrently to our work, a new approach for anonymizing transactional data was proposed in [12]. The authors introduce a top-down local recoding method, named *Partition* using the same assumptions for data as we do, but they focus on a stronger privacy guaranty, the complete k-anonymity. They present a comparison between their method and the AA algorithm we present in Section 4.4. *Partition* is superior to AA, when large values of m are used by AA to approximate complete k-anonymity[1], The comparison favors *Partition* when using a large m to approximate complete k-anonymity, but it is inferior to AA for a small m. Since both local recoding algorithms we propose in this paper use the AA algorithm, we are able to provide a qualitative comparison with the *Partition* algorithm. The comparison appears in Section 6 and it is based on the experimental results of [12] for the AA and *Partition* algorithms.

3 Problem Setting

Let D be a database containing $|D|$ transactions. Each transaction $t \in D$ is a *set* of items.[2] Formally, t is a non-empty subset of $\mathcal{I} = \{o_1, o_2, \dots, o_{|\mathcal{I}|}\}$. $\mathcal{I}$ is the *domain* of possible items that can appear in a transaction. We assume that the database provides answers to subset queries, i.e., queries of the form $\{t \mid (qs \subseteq t) \wedge (t \in D)\}$, where qs is a set of items from $\mathcal{I}$ provided by the user. The number of query items provided by the user in qs defines the size of the query. We define a database as k^m-anonymous as follows:

[1] It should be noted that even if m is equal to the maximum record length, k^m-anonymity is not equivalent to k-anonymity, as explained in [12].

[2] We consider only sets and not multisets for reasons of simplicity. In a multiset transaction, each item is tagged with a number of occurrences, adding to dimensionality of the solution space. We can transform multisets to sets by considering each combination of (⟨item⟩,⟨number of appearances⟩) as a different item.

Definition 1 Given a database D, no attacker that has background knowledge of up to m items of a transaction $t \in D$ can use these items to identify less than k tuples from D.

In other words, any subset query of size m or less, issued by the attacker should return either nothing or more than k answers. Note that queries with zero answers are also secure, since they correspond to background information that cannot be associated to any transaction.

3.1 Generalization Model

If D is not k^m-anonymous, we can transform it to a k^m-anonymous database D' by using generalization. Generalization refers to the mapping of values from an initial domain to another domain, such that several different values of the initial domain are mapped to a single value in the destination domain. In the general case, we assume the existence of a generalization hierarchy where each value of the initial domain can be mapped to a value in the next most general level, and these values can be mapped to even more general ones, etc. For example, we could generalize items "skimmed milk" "choco-milk", and "full-fat milk", to a single value "milk" that represents all three detailed concepts. At a higher generalization level, "milk", "yogurt", and "cheese", could be generalized to "dairy product". The effect of generalization is that sets of items which are different in a detailed level (e.g., {skimmed milk, bread}, {full-fat milk, bread}) could become identical (e.g., {milk, bread}).

Formally, we use a generalization hierarchy for the complete domain $\mathcal{I}$ of items that may appear in a transaction. Such an exemplary hierarchy is shown in Figure 1. In this example we assume $\mathcal{I} = \{a_1, a_2, b_1, b_2\}$, items a_1, a_2 can be generalized to A, items b_1, b_2 can be generalized to B, and the two classes A, B can be further generalized to ALL.

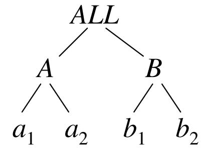


Fig. 1 Sample generalization hierarchy

If a generalization is applied to a transaction, this leads to the replacement of some original items in the transaction by generalized values. For example, the generalization rule $\{a_1, a_2\} \rightarrow A$, if applied to all transactions of the database D, shown in Figure 2a, will result to the database D', shown in Figure 2b. Notice that we consider strict set semantics for the transactions; this leads to possibly reduced cardinality for the generalized sets. For example, t_4 is transformed to t'_4 which has two items instead of three. We say that itemset t'_4 is a generalization of itemset t_4. Formally a generalized itemset is defined as follows:

Definition 2 A itemset gt is a generalization of itemset t iff $\forall o (o \in t) \Leftrightarrow ((o \in gt) \vee (g(o) \in gt))$

Where $g(o)$ stands for a generalization of item o.

id	contents
t_1	$\{a_1, b_1, b_2\}$
t_2	$\{a_2, b_1\}$
t_3	$\{a_2, b_1, b_2\}$
t_4	$\{a_1, a_2, b_2\}$

(a) original database (D)

id	contents
t'_1	$\{A, b_1, b_2\}$
t'_2	$\{A, b_1\}$
t'_3	$\{A, b_1, b_2\}$
t'_4	$\{A, b_2\}$

(b) transformed database (D')

Fig. 2 Transformation using $\{a_1, a_2\} \rightarrow A$

If we aim for 2^2-anonymity, database D in Figure 2a is not secure, since there are 2-itemsets (e.g., $\{a_1, b_1\}$) that appear in less than $k = 2$ transactions (e.g., only in t_1). The application of the generalization rule $\{a_1, a_2\} \rightarrow A$ to all transactions of D results in a 2^2-anonymous database D' (shown in Figure 2b). To test the anonymity requirement, we have to translate all possible 2-itemsets from the original domain, to the generalized domain and count their supports in D'. For example, finding the support of $\{a_1, b_2\}$ in D' is equivalent to finding the support of $\{A, b_2\}$ in D', which is 3 ($\geq k$). Notice that, when testing an original itemset containing two or more items that are mapped to the same generalized value, this translates to testing a lower-cardinality set. For example, $\{a_1, a_2\}$ is generalized to $\{A\}$ in D', which has a support of 4. Seeing A in the published dataset one can only infer that $\{a_1, a_2\}$ appears up to 4 times in the original dataset.

All the proposed algorithms, except from the LRA, have a *global recoding* approach [5,14] which applies the selected generalization rules to *all* transactions in the database. An example of global recoding has already been shown in Figure 2. The LRA algorithm uses an alternative *local recoding* generalization [15,2] would apply selected generalization rules to a subset of the transactions and result in a database where items are generalized at different levels at different transactions. This allows more flexibility, and facilitates partitioning of the anonymization procedure.

3.2 Possible solutions

A transformation of the original database D to a D' is related to a set of generalizations that apply to domain $\mathcal{I}$. Formally, the set of possible transformations corresponds to the set of possible horizontal cuts of the hierarchy tree. Each such cut, defines a unique set of generalization rules. Figure 3 shows the possible cuts of the hierarchy depicted in Figure 1 together with the corresponding generalization rules. Each cut corresponds to a set of non-overlapping subtrees of the hierarchy, which altogether span the complete domain $\mathcal{I}$. The root of each subtree correspond to the generalization rule which maps all values in its leaves to the label of the root.

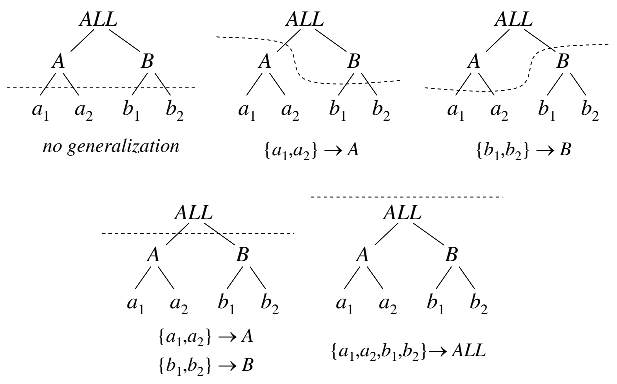


Fig. 3 Possible domain generalizations

The trivial generalization $\mathcal{I} \rightarrow ALL$, suppresses the whole database, since all items are replaced by a generalized value (e.g., "product"). This generalization always leads to a k^m-anonymous database, assuming that the original database D has at least k transactions. However, the transformation eliminates all information from the database, making it useless. In Section 3.3 we formally define the information loss of a generalization rule (and a hierarchy cut).

The set of possible cuts also form a hierarchy, based on the generalizations implied by them. Figure 4 shows this hierarchy lattice for the cuts of Figure 3. We say that cut c_1 is a generalization of cut c_2, denoted by $c_1 \succ c_2$, if the rules of c_1 generalize the rules of c_2. For example, in Figure 4, cut $\langle\{a_1, a_2, b_1, b_2\} \rightarrow ALL\rangle$ is a generalization of cut $\langle\{a_1, a_2\} \rightarrow A\rangle$. A cut can also be denoted by the generalization it derives; e.g., cut $\langle\{a_1, a_2\} \rightarrow A\rangle$ can be denoted as $\langle A, b_1, b_2\rangle$.

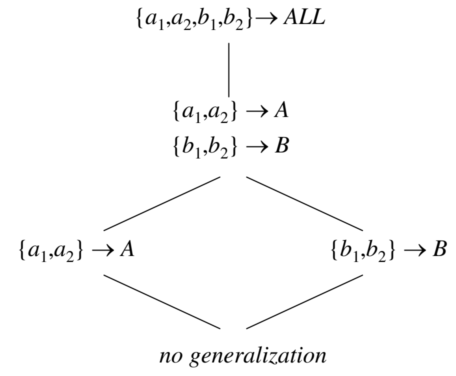


Fig. 4 Hierarchy of domain generalizations

3.3 Information loss

All privacy-preserving transformations cause information loss, which must be minimized in order to maintain the ability to extract meaningful information from the published data. A variety of information loss metrics have been proposed. The *Classification Metric* (CM) [13] is suitable when the purpose of the anonymized data is to train a classifier, whereas the *Discernibility Metric* (DM) [5] measures the cardinality of the anonymized groups. More accurate is the *Generalized Loss Metric* [13] and the similar *Normalized Certainty Penalty* (NCP) [29]. In the case of categorical attributes NCP is defined with respect to the hierarchy. Let p be an item in $\mathcal{I}$. Then:

$$NCP(p) = \begin{cases} 0, & |u_p| = 1 \\ |u_p|/|\mathcal{I}|, & otherwise \end{cases}$$

where u_p is the node of the item generalization hierarchy where p is generalized. $|u_p|$ and $|\mathcal{I}|$ are the number of leaves under u_p and in the entire hierarchy, respectively. Intuitively, the NCP tries to capture the degree of generalization of each item, by considering the ratio of the total items in the domain that are indistinguishable from it. For example, in the hierarchy of Figure 1, if a_1 is generalized to A in a transaction t, the information loss $NCP(a_1)$ is 2/4. The NCP for the whole database weights the information loss of each generalized item using the ratio of the item appearances that are affected to the total items in the database. If the total number of occurrences of item p in the database is C_p, then the information loss in the whole database due to the generalization can be expressed by:

$$NCP(D) = \frac{\sum_{p \in \mathcal{I}} C_p \cdot NPC(p)}{\sum_{p \in \mathcal{I}} C_p}$$

The information loss of a particular generalization (cut) ranges from 0 to 1 and can be easily measured.

If we scan the database D once and bookkeep the supports of all items in $\mathcal{I}$, we can use this information to measure the information loss of any generalization, without having to access the database again. For example the information loss due to cut $\langle\{a_1, a_2\} \rightarrow A\rangle$ in the database of Figure 2a is $\frac{2\cdot 0.5+3\cdot 0.5+0+0}{11} = \frac{2.5}{11}$.

In addition, to NCP, we introduce an additional (more specialized) information loss measure, which reflects the impact of anonymization to the result of data mining on the anonymized data. The *multiple level mining loss* (ML^2) measure expresses the information loss in the detection of multi-level frequent itemsets, when mining is performed on the published dataset instead of the original one. Mining a dataset at multiple levels of a generalization hierarchy is a technique that has attracted interest in the data mining literature [9, 10], since it allows detecting frequent association rules and frequent itemsets that might not appear in the most detailed level of the data. Assume for example that we have a dataset that describes sales of a supermarket. When using the original data, which contain items at the most detailed product level, we might not detect any interesting frequent itemset that associates a specific skimmed milk product with a specific type of cookie. Still, if we generalize the data (or move to a higher level of abstraction in the terminology of [10]), we might be able to infer that 'skimmed milk' and 'cookies' appear frequently together. When using a generalization-based method to anonymize the original data, frequent itemsets at the lower levels of the hierarchy might be hidden and the correlations between the items can only be detected at higher levels. ML^2 expresses the percentage of all frequent itemsets that do not appear in the appropriate generalization level when a multilevel mining algorithm is applied to the published dataset instead of the original one. More formally:

$$ML^2 = 1 - \frac{\sum_i^h FI_i(D_p)}{\sum_i^h FI_i(D)} \tag{1}$$

where $FI_i()$ returns the number of frequent itemsets at generalization level i, and D_p and D stand for the published and original dataset respectively.

We can further enhance the insight on the information loss in multiple level data mining by tracing the difference between the generalization level on which a frequent item appears in the original data and the generalization level it first appears in the published data. Assume, for example, that in the dataset with supermarket sales, one can infer that 'skimmed milk 2%' and 'cookies 100g' appear together in some hierarchy level l. Since such a frequent itemset appears in level l, a more generalized form of the itemset should appear in all levels $l' > l$. In the anonymized dataset generalizations can possibly hide 'skimmed milk 2%' or 'cookies 100g', so an itemset that expresses this correlation might firstly appear in level $l + d$ as {'skimmed milk', 'cookies'}. The difference d between level l where a frequent itemset appears in the original data and level $l + d$, which is the lowest level at which the same correlation is expressed in the published dataset can be used to define the the *differential multiple level mining loss* (dML^2). More formally:

$$dML^2 = \frac{\sum_i^h \sum_{fi \in \mathcal{FI}_i(D)} dlevel(fi, D, D_p)}{\sum_i^h FI_i(D)} \tag{2}$$

where $dlevel(fi, D, D_p)$ returns the level difference between the generalization level at which a frequent itemset fi first appears in the original data D, and the generalization level at which the same frequent itemset or a more generalized form of it first appears in the published data D_p. $\mathcal{FI}_i(D)$ is the set of all frequent itemsets at level i in D and $FI_i(D)$ is their total number. Note, that the generalized form of a frequent itemset might have a smaller arity than the original, if some of the items have been generalized to a common ancestor.

In the rest of the paper, unless otherwise stated, we will use the generic NCP metric when referring to information loss. ML^2 and dML^2 are only used in our experimental evaluation to assess the quality of our anonymization techniques from a different prism.

3.4 Monotonicity

We now provide a property (trivial to prove), which is very useful towards the design of algorithms that seek for the best hierarchy cut.

Property 1 (Monotonicity of cuts) If the hierarchy cut c results in a k^m-anonymous database D' then all cuts c', such that $c' \succ c$ also result in a k^m-anonymous database D'.

In addition, we know that if $c' \succ c$, then c' has higher cost (information loss) than c. Based on this and Property 1, we know that as soon as we find a cut c that qualifies the k^m-anonymity constraint, we do not have to seek for a better cut in c's ancestors (according to the cut hierarchy). Therefore, for a database with at least k transactions, there is a set C of cuts, such that for each $c \in C$, (i) the k^m-anonymity constraint is satisfied by the database D' resulting after applying the generalizations in c to D, and (ii) there is no descendant of c in the hierarchy of cuts, for which condition (i) is true.

We call C the set of *minimal* cuts. The ultimate objective of an anonymization algorithm is to find the *optimal* cut c_{opt} in C which incurs the minimum information loss by transforming D to D'. In the next section, we propose a set of algorithms that operate in this direction.

4 Anonymization techniques

The aim of the anonymization procedure is to detect the cut in the generalization hierarchy that prevents any privacy breach and at the same time it introduces the minimum information loss. We first apply a systematic search algorithm, which seeks for the optimal cut, operating in a similar fashion like Incognito [14]. This algorithm suffers from the dimensionality of the generalization space and becomes unacceptably slow for large item domains $\mathcal{I}$ and generalization hierarchies. To deal with this problem we propose heuristics, which instead of searching the whole generalization space, they detect the privacy breaches and search for local solutions. The result of these methods is a cut on the generalization hierarchy, which guarantees k^m-anonymity, while incurring low information loss. Before presenting these methods in detail, we present a data structure, which is used by all algorithms to accelerate the search of itemset supports.

4.1 The count-tree

An important issue in determining whether applying a generalization to D can provide k^m-anonymity or not is to be able to count efficiently the supports of all the combinations of m items that appear in the database. Moreover, if we want to avoid scanning the database each time we need to check a generalization, we must keep track of how each possible generalization can affect the database. To achieve both goals we construct a data structure that keeps track not only of all combinations of m items from $\mathcal{I}$ but also all combinations of items from the generalized databases that could be generated by applying any of the possible cuts in the hierarchy tree. The information we trace is the support of each combination of m items from $\mathcal{I}$, be detailed or generalized. Note, that if we keep track of the support of all combinations of size m of items from $\mathcal{I}$, it is enough to establish whether there is a privacy breach or not by shorter itemsets. This follows from the Apriori principle that states that the support of an itemset is always less or equal than the support of its subsets.

To count the supports of all these combinations and store them in a compressed way in our main memory, we use a count-tree data structure, similar to the FP-tree of [11]. An exemplary such tree for the database of Figure 2a and $m = 2$ is shown in Figure 5. The tree assumes an order of the items and their generalizations, based on their frequencies (supports) in D. For instance, the (decreasing frequency) order of (generalized) items in the database of Figure 2a is $\{ALL, A, B, a_2, b_1, b_2, a_1\}$. To compute this order, a database scan is required. If we want to avoid the scan, a heuristic approach is to put the items in order of the number of detailed items they generalize (e.g., $\{ALL, A, B, a_1, a_2, b_1, b_2\}$). However, since it makes no sense to count the number of ALL occurrences (they are always $|D|$) and ALL cannot be combined with any other item (it subsumes all items), there is no need to include ALL in the tree. The support of each itemset with up to m items can be computed by following the corresponding path in the tree and using the value of the corresponding node. For example, the support of itemset $\{a_1\}$ is 2 (i.e., the value of node a_1) and the support of itemset $\{A, b_2\}$ is 3 (i.e., the value at the node where path $A \rightarrow b_2$ ends).

Algorithm 1 is a pseudocode for creating this tree. The database is scanned and each transaction is expanded by adding all the generalized items that can be derived by generalizing the detailed items. Since we assume strict set semantics, each generalized item appears only once. The expansion of the database of Figure 2a is shown in Figure 6. Note that the expansion does not have to be done explicitly for the database; in practice, we expand each transaction t read from D on-the-fly. For each expanded transaction t, we find all subsets of t up to size m which do not contain any two items i, j, such that i is a generalization of j, we follow the corresponding path at the tree (or create it if not already there) and increase the counter of the final node in this search. For example, the expanded transaction t_2 in Figure 6 generates the following itemsets: a_2, b_1, A, B, $\{a_2, b_1\}$, $\{a_2, B\}$, $\{b_1, A\}$, $\{A, B\}$.

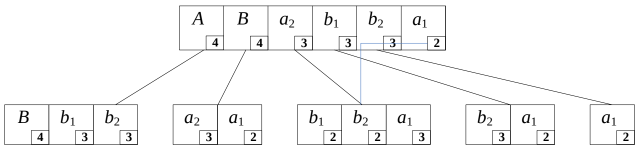


Fig. 5 Count-tree for the database of Figure 2

Although this tree facilitates fast support counting for any ad-hoc set of m items or less, generalized in any ad-hoc fashion, (i) it requires a large amount of memory, and (ii) its construction incurs high computational

id	contents
t_1	$\{a_1, b_1, b_2, A, B\}$
t_2	$\{a_2, b_1, A, B\}$
t_3	$\{a_2, b_1, b_2, A, B\}$
t_4	$\{a_1, a_2, b_2, A, B\}$

Fig. 6 Expanded Database

Algorithm 1 Creation of the tree for k^m anonymity

```
    *populateTree*($D$, $tree$, $m$)
1: **for all** $t$ in $D$ **do**                              ▷ for each transaction
2:     expand $t$ with the supported generalized items
3:     **for all** combination of $c \leq m$ items in the expanded $t$ **do**
4:         **if** $\nexists i, j \in c$ such that $i$ generalizes $j$ **then**
5:             insert $c$ in $tree$
6:             increase the support counter of the final node
```

cost, as all m-sized or less itemsets, generalized or not, in every transaction t have to be found and inserted into the tree. For a database of $|D|$ transactions, each of size τ, the tree construction cost is $O(|D| \cdot \binom{\tau}{m})$. Our best algorithm, discussed in Section 4.4 greatly decreases this cost, by examining the itemsets level-by-level and using the Apriori property to reduce the number of item combinations that need to be inserted to the tree and counted.

4.2 Optimal anonymization

To find the optimal cut, i.e., the generalization that satisfies k^m-anonymity and has the least information loss, we can examine systematically the generalizations in the cut hierarchy, in a bottom-up, breadth-first fashion. Initially, the cut c_{ng} which corresponds to no generalization (i.e., bottommost cut in the hierarchy) is put to a queue Q. While Q is not empty, we remove the first cut c from it and examine whether c satisfies k^m-anonymity. If so, it becomes a candidate solution. Any immediate ancestors of c are marked as non-optimal (since they cannot be minimal cuts) and removed from Q if they appear there. The marked combinations are kept in a hash table H, so they will not be added again in the in Q at the future. The information loss of c is computed and compared with that of the best found cut c_{opt} so far. If c's information loss is lower, then c replaces c_{opt} as the optimal solution found so far.

If c does not satisfy k^m-anonymity, its immediate ancestors in the hierarchy, which do not have a descendant cut that satisfies k^m-anonymity, are added to the queue of cuts Q to be examined at the next lattice levels. The algorithm terminates as soon as Q is empty. Algorithm 2 is a pseudocode of this *optimal anonymization* (OA) algorithm.

Note that the immediate ancestors of a cut c are created constructively, by replacing a set of (generalized) items which have common parent in the item hierarchy, by their parent. For example, cut $\langle A, B\rangle$ is derived from $\langle A, b_1, b_2\rangle$, by replacing $\{b_1, b_2\}$, by B. This way the complete hierarchy lattice of the cuts does not need to be precomputed.

It is easy to see that when the size of $\mathcal{I}$ (and the corresponding generalization hierarchy of items) grows the algorithm becomes prohibitive expensive. In specific, assuming that the item generalizations form a tree hierarchy of node degree κ, then the number of possible cuts is the solution to the recurrence $T(N) = 1 + T(N/\kappa)^\kappa$, for $N = \mathcal{I}$, which is lower-bounded by $2^{N/\kappa}$, i.e., exponential to N. Moreover, each iteration requires checking the supports of all m-itemsets with respect to the corresponding generalization, in order to determine whether the current node satisfies k^m-anonymity or not. The basic problem of the optimal algorithm is that it performs its search based on the domain of the database. In the next sections we present heuristic approaches that greedily search for a domain generalization that provides k^m-anonymity to D and, at the same time, have low information loss.

Algorithm 2 Optimal Anonymization algorithm

```
    *OA*($D$, $\mathcal{I}$, $k$, $m$)
1: $c_{opt}$ := null; $c_{opt}.cost$ := $\infty$            ▷ initialize $c_{opt}$
2: add $c_{ng}$ to an initially empty queue $Q$
3: **while** ($Q$ is not empty) **do**
4:     pop next cut $c$ from $Q$
5:     **if** $c$ does not provide $k^m$-anonymity to $D$ **then**
6:         **for all** immediate ancestors $c_{ans}$ of $c$ **do**
7:             **if** $c_{ans}$ does not appear in $H$ **then**
8:                 push $c_{ans}$ to $Q$
9:     **else**                          ▷ $c$ provides $k^m$-anonymity to $D$
10:        **for all** immediate ancestors $c_{ans}$ of $c$ **do**
11:            add $c_{ans}$ to $H$
12:            **if** $c_{ans}$ in $Q$ **then**
13:                delete $c_{ans}$ from $Q$
14:        **if** $c.cost < c_{opt}.cost$ **then**
15:            $c_{opt}$ := $c$
16: **return** $c_{opt}$
```

4.3 Direct anonymization

The basic idea of our first heuristic algorithm, called *direct anonymization* (DA), is to scan the count-tree once for possible privacy breaches and then use the generalized combinations to track down a solution that solves each problem. Similar to the optimal anonymization (OA) algorithm, this method is based on the pre-computation of the complete count-tree for sets consisting of up to m (generalized) itemsets. DA scans the tree to detect m-sized paths that have support less than k. For each such path, it generates all the possible generalizations, in a level-wise fashion, similar to the optimal anonymization (OA) algorithm, and finds among the minimal cuts that solve the specific problem, the one which incurs the lowest information loss.

Specifically, once the count-tree has been created, DA initializes the output generalization c_{out}, as the bottommost cut of the lattice (i.e., no generalization). It then performs a preorder (i.e., depth-first) traversal of the tree. For every node encountered (corresponding to a prefix), if the item corresponding to that node has already been generalized in c_{out}, DA backtracks, as all complete m-sized paths passing from there correspond to itemsets that will not appear in the generalized database based on c_{out} (and therefore their supports need not be checked). For example, if the algorithm reaches the prefix path B-a_2 the algorithm will examine its descendants only if B and a_2 have not already been further generalized. Note that this path will be examined even if the items b_1 and b_2 have not been generalized to item B. Due to the monotonicity of the problem, we know that if B-a_2 leads to a privacy breach, then it is certain that b_1-a_2 and b_1-a_1 lead to privacy breach. Addressing the problem for the path B-a_2 allows the algorithm to avoid examining the other two paths. During the traversal, if a leaf node is encountered, corresponding to an m-itemset J (with generalized components or not), DA checks whether the corresponding count $J.count$ is less than k. In this case, DA seeks for a cut which (i) includes the current generalization rules in c_{out} and (ii) makes the support of J at least k. This is done in a similar fashion as in OA, but restricted only to the generalization rules which affect the items of J. For example, if $J = \{a_1, a_2\}$, only generalizations $\{a_1, a_2\} \to A$ and $\{a_1, a_2, b_1, b_2\} \to ALL$ will be tested. In addition, from the possible set of cuts that solve the anonymity breach with respect to J, the one with the minimum information loss is selected (e.g., $\{a_1, a_2\} \to A$). The generalization rules included in this cut are then *committed* (i.e., added to c_{out}) and any path of the count-tree which contains items at a more detailed level (e.g., a_1 and a_2) than c_{out} is pruned from search subsequently.

Algorithm 3 is a high-level pseudocode for DA.

As an example, assume that we want to make the database of Figure 2a 2^2-anonymous. First, DA constructs the count-tree, shown in Figure 5. c_{out} is initialized to contain no generalization rules. Then DA performs a preorder traversal of the tree. The first leaf node encountered with a support less than 2 is a_1 (i.e., path a_2-a_1). The only minimal cut that makes $\{a_2, a_1\}$ k-anonymous is $\{a_1, a_2\} \to A$, therefore the corresponding rule is added to c_{out}. DA then backtracks to the next entry of the root (i.e., b_1) since any other path starting from a_2 would be invalid based on c_{out} (i.e., its corresponding itemset could not appear in the generalized database according to c_{out}). The next path to check would be b_1-b_2, which is found non-problematic. DA then examines b_1-a_1, but backtracks immediately, since a_1 has already been generalized in c_{out}. The same happens with b_2-a_1 and the algorithm terminates with output the cut $\langle\{a_1, a_2\} \to A\rangle$.

Algorithm 3 Direct Anonymization

```
    DA ($D$, $\mathcal{I}$, $k$, $m$)
 1: scan $D$ and create count-tree
 2: initialize $c_{out}$
 3: for each node $v$ in preorder count-tree traversal do
 4:     if the item of $v$ has been generalized in $c_{out}$ then
 5:         backtrack
 6:     if $v$ is a leaf node and $v.count < k$ then
 7:         $J$ := itemset corresponding to $v$
 8:         find generalization of items in $J$ that make $J$ $k$-anonymous
 9:         merge generalization rules with $c_{out}$
10:         backtrack to longest prefix of path $J$, wherein no item has been generalized in $c_{out}$
11: return $c_{out}$
```

The main problem of DA is that it has significant memory requirements and computational cost, because it generates and scans the complete count-tree for all m-sized combinations, whereas it might be evident from smaller-sized combinations that several generalizations are necessary. This observation leads to our next algorithm.

4.4 Apriori-based anonymization

Our second heuristic algorithm is based on the apriori principle; if an itemset J of size i causes a privacy breach, then each superset of J causes a privacy breach. Thus, it is possible to perform the necessary generalizations progressively. First we examine the privacy breaches that might be feasible if the adversary knows only one item from each transaction, then two and so forth till we examine privacy threats from an adversary that knows m items.

The benefit of this algorithm is that we can exploit the generalizations performed in step i, to reduce the search space at step i+1. The algorithm practically iterates the direct algorithm for combinations of sizes $i = \{1, \ldots, m\}$. At each iteration i the database is scanned and the count-tree is populated with itemsets of length i. The population of the tree takes into account the current set of generalization rules c_{out}, thus significantly limiting the combinations of items to be inserted to the tree. In other words, in the count-tree at level i, i-itemsets which contain items already generalized in c_{out} are disregarded. Algorithm 4 is a pseudocode for this *apriori-based anonymization* (AA) technique.

Algorithm 4 Apriori-based Anonymization

```
AA ($D$, $\mathcal{I}$, $k$, $m$)
1: initialize $c_{out}$
2: for $i := 1$ to $m$ do                          ▷ for each itemset length
3:     initialize a new count-tree
4:     for all $t \in D$ do                        ▷ scan $D$
5:         extend $t$ according to $c_{out}$
6:         add all $i$-subsets of extended $t$ to count-tree
7:     run DA on count-tree for $m = i$ and update $c_{out}$
```

Note that in Line 5 of the algorithm, the current transaction t is first expanded to include all item generalizations (as discussed in Section 4.1), and then all items that are generalized in c_{out} are *removed* from the extended t. For example, assume that after the first loop (i.e., after examining 1-itemsets), $c_{out} = \langle\{a_1, a_2\} \rightarrow A\rangle$. In the second loop ($i$=2), $t_4 = \{a_1, a_2, b_2\}$ is first expanded to $t_4 = \{a_1, a_2, b_2, A, B\}$ and then reduced to $t_4 = \{b_2, A, B\}$, since items a_1 and a_2 have already been generalized in c_{out}. Therefore, the 2-itemsets to be inserted to the count-tree due to this transaction are significantly decreased.

The size of the tree itself is accordingly decreased since combinations that include detailed items (based on c_{out}) do not appear in the tree. As the algorithm progresses to larger values of i, the effect of pruned detailed items (due to c_{out}) increases because (i) more rules are expected to be in c_{out} (ii) the total number of i-combinations increases exponentially with i. Overall, although D is scanned by AA m times, the computational cost of the algorithm is expected to be much lower compared to DA (and OA). The three algorithms are compared in the next section with respect to (i) their computational cost and (ii) the quality of the k^m-anonymization they achieve.

5 Negative knowledge and ℓ-diversity

In this section, we discuss the impact of negative knowledge in k-anonymity. In addition, we outline the reasons that make applying the concept of ℓ-diversity hard to apply in our privacy problem and provide an ℓ-diversity definition for cases where such a concept can be applied.

5.1 Negative knowledge

An important issue in privacy preservation is the treatment of *negative knowledge*: the fact that an attacker knows that a record does *not* contain a certain item. This problem is more evident in multidimensional data, since in relational data nulls in quasi identifiers are usually treated as all other values. Negative knowledge acting as an *inference channel* for guessing sensitive values is studied in [4]: knowing that a value does not exist in a record can be exploited to breach privacy in several cases. Negative knowledge can be used in the context of this paper to isolate fewer than k-tuples for a given combination of m or less items, still we have opted to focus on positive knowledge for three reasons:

1. Taking into account negative knowledge in a traditional privacy guaranty, like k-anonymity, will lead to extensive information loss due to the curse of dimensionality [1]. Forfeiting negative knowledge was one of the mechanisms we used to reduce this cost. We believe it offers a good trade-off since negative knowledge is fairly weak. Transactional data are usually sparse, containing only a small subset of the itemset domain. This means that positively knowing an item that appears in a transaction is a lot more identifying than knowing an item that does not.
2. Apart from being weaker, negative knowledge is usually a lot harder to obtain in most practical scenarios. E.g., in a supermarket a customer usually buys tenths of different products and never buys thousands of them. Not only is the knowledge of those she does not buy unimportant compared to the knowledge of those that she buys, but also is it a lot harder to obtain this knowlegde. For a casual observer she has to follow all visits of the customer to the supermarket in order to make sure that a product is not bought, but she only has to observe the customer for a limited time to know the buying of certain products. The same goes for more powerful attackers who might be able to trace the consumption of whole product series (for example through promotional campaigns for certain products, that require users to register); it is easier to acknowledge that someone exploited the promotion and bought a product than to rule out that someone has not bought it.
3. Finally, we have left negative knowledge out of the scope of our discussion for reasons of simplicity. The k^m-anonymity guaranty can easily be extended to take into account negative knowledge. Instead of considering m-sized combinations of items that appear in a record, we have to consider any m-sized

combination of items whether they appear in a record or not. To make this more clear, assume that we populate each record with the negation of each product that does not appear in it; e.g., for a user that has bought only products a and b, we should mark $\{a, b, \neg c, \neg d\}$ (assuming that our domain contains only items a, b, c, d). Using such a representation we can generalize our privacy guarantee so that the anonymized dataset will consider all m-combinations of any positively or negatively traced product in the records. In other words, any combination of m items that are associated or not associated with a person, will appear at least k times in the anonymized database. For instance, for $m = 3$ $\{a, b, \neg c\}$ can act as quasi identifier. For a dataset to be k^3 anonymous, this quasi identifier has to appear at least k times.

5.2 ℓ^m-diversity

An important weakness of the k-anonymity guaranty is that it does not prohibit an adversary from uncovering certain values that are associated to a person. For example assume that a database D contains three records that are all identical: a, b, s_1, where a and b are quasi identifiers and s_1 a sensitive value. The database might be 3-anonymous, but an adversary that knows a person which is associated to values a and b can be certain that the same person is associated with value s_1 too. To tackle this problem the authors of [17], proposed the ℓ-diversity guaranty where any person cannot be associated with less than ℓ sensitive values. ℓ-diversity practically leads to the creation of equivalence classes, where each set of quasi identifiers is associated with at least ℓ different sensitive values. This way linking public knowledge (quasi identifiers) to sensitive knowledge cannot be done with certainty. There are basically three problems with applying ℓ-diversity in our context:

1. Without the distinction between sensitive and non-sensitive values, the ℓ-diversity guaranty implies a requirement contradictory to almost all utility metrics. While utility metrics try to preserve the statistical properties and the correlation between items, ℓ-diversity requires that no itemset can be associated with any other itemset with probability higher than $1/\ell$. Practically the dataset would be rendered useless if all data were to be treated both as sensitive and as quasi identifiers. ℓ-diversity is applicable only if are able to clearly differentiate between items that are quasi identifiers and items that are sensitive.
2. In most practical applications it would be hard to characterize as sensitive and non-sensitive the items that might appear in a dataset. While in the relational context only few attributes have to be characterized, here we must have this knowledge for each item in a large domain.
3. Even if we knew which items are sensitive and which are not, sensitive items can still act as quasi identifiers. For example if sensitive items s_1, s_2, s_3 appear in a single record, in many practical scenarios we cannot rule out the possibility that an adversary has somehow managed to associate s_1 and s_2 with a specific person. Then she would be able to use these sensitive items to associate s_3 with the same person.

Despite the aforementioned problems, there can still be cases where a practical distinction can be made between sensitive values and quasi identifiers. The distinction also implies that sensitive values cannot act as quasi identifiers, i.e., there is no guarantee if the adversary manages somehow to learn some of the sensitive values. In this setting we can easily adapt the k^m-anonymity concept to ℓ^m-diversity.

Definition 3 Given a database D where all items are either quasi identifiers or sensitive values, no attacker that has background knowledge of up to m quasi identifier items of a transaction $t \in D$ can associate these items with *any* combination of sensitive values with probability $1/\ell$.

The difference of ℓ^m-diversity from k^m anonymity is that ℓ^m-diversity considers only a subset of D as quasi identifiers and has a different privacy criterion when checking if a combination of quasi identifiers can lead to a privacy breach. Adjusting our algorithms to guarantee ℓ^m-diversity is rather straight-forward. We now only have to consider combinations of items that can act as quasi-identifiers, thus the count tree has to be built solely on them. Instead of using the support of a combination c, $s(c)$ as a criterion for privacy breaches we have to also use the support of the most frequent sensitive itemset s, $sup(s)$ that co-appears with c. If $\frac{sup(s)}{sup(c)} > \frac{1}{l}$ then there is a privacy breach and items from c have to be generalized. The criterion for choosing the items remains the same: we choose the generalization that solves the problem and has the least cost. The problem is solved if the resulting combination c' has a ratio $\frac{sup(s')}{sup(c')} \leq \frac{1}{l}$, where s' is the most frequent combination of sensitive items that appears in the records that contain c'. Since only quasi identifiers get generalized, the support of their combinations rises (whereas the support of sensitive items in D remains stable), thus ultimately ℓ^m-diversity will be achieved.

5.3 Local recoding

Local recoding has been used in traditional k-anonymity and ℓ-diversity problems. It relies on generalizing existing values, but unlike global recoding, where all or none of the occurrences of an item o are replaced with a more generalized item g, the replacement here can be partial; only some occurrences of o are replaced with g. Local recoding can potentially reduce the distortion in the data, by replacing o only in a *neighborhood* of the data. The detection of a suitable neighborhood is hard to achieve in a multidimensional setting, where it is hard to detect data with similar values. Still, by using local recoding we can process the database *locally*, i.e., only a part of the dataset is processed at each moment. Even if the information loss is greater compared to the AA algorithm, there is better potential for scalability, since the database can be partitioned according to the available memory.

As we discussed, the memory requirements for the count tree that is used by the anonymization heuristics can be significant. Algorithms like AA algorithm reduce the count tree's size, by progressively computing the required generalizations for publishing a k^i-anonymous, $i = 1, \ldots, m$, database. Nevertheless, even AA does not provide effective control over the memory requirements of the anonymization procedure. If the database and the items domain are large, AA might have requirements that exceed the available memory.

The basic idea in the local recoding anonymization (LRA) algorithm is to partition the original dataset, and anonymize each part independently, using one of the proposed algorithms (e.g., AA). It is easy to see that if all parts of the database are k^m-anonymous, then the database will be k^m-anonymous too. If a part is k^m anonymous, then each m-sized combination of items would appear at least k times in this part. Unifying this part with other database parts, cannot reduce the number of appearances of any m-sized combination. Hence, if all parts are k^m-anonymous, i.e. all m-sized combinations of items appear at least k times in each part, then the whole anonymized database (which is their union) every m-sized combination will appear at least k-times. By splitting the database into the desired number of chunks, we can implicitly define the memory requirements.

Local anonymization can affect the data quality in two, conflicting, ways. On the one hand, it fixes a problem locally, thus it does not distort the whole database in order to tackle a privacy threat. For example, assume that we split the database in two parts D_1 and D_2 and that items o_1, o_2, which are both specializations of item g, appear 2 times each in D_1 and 0 and 4 times in D_2, respectively. With $k = 3$, local anonymization would generalize both o_1 and o_2 to g in D_1, and it would leave o_2 intact in D_2. On the other hand, the algorithm decides whether there is a privacy breach at a local level, and fails to see whether this problem holds when the whole database is considered. For example, if we decide to split database D to two parts D_1 and D_2, a combination c which appears $2k-2$ times ($k > 2$) and does not pose any threat for the user privacy, might only appear $k-1$ times in both D_1 and D_2, thus causing generalizations that could have been avoided. Intuitively, we could mitigate this negative effect by mustering all the occurrences of any item combination in a single chunk of D. This is far from trivial to achieve, since each record supports numerous different combinations. Splitting the database in a way that we get both the desired number of chunks and completely different item combinations at each chunk is usually impossible. In the following we propose a heuristic that tries to split the database into the desired number of chunks, by minimizing the overlap of the sets of different combinations of items that appear in each chunk.

Decimal	**Gray**	**Binary**
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

Table 1 Decimal, binary and gray coding of numbers

5.3.1 Partitioning by Gray code ordering

The Gray code [23] is a binary numeral system where two successive values differ in only one digit (see Table 1). Numbers that are relatively close will have relatively similar Gray codings in terms of the placement of 0s and 1s in their representation. In the information theory terminology two successive values in the Gray code system will have a Hamming distance of 1.[3] The Hamming distance between two strings or sequences of equal length is the number of positions where the corresponding symbols are different.

We exploit the gray code ordering in the following way: For each item i of the domain I, we create a signature with $|I|$ bits, where the i-th bit is 1 and all others are 0. Then, for each transaction of t of the

[3] Assuming that all numbers are padded with 0s to have an equal length representation

a_1	0001
a_2	0010
b_1	0100
b_2	1000

id	contents	gray code	decimal
t_1	$\{a_1, b_1, b_2\}$	1101	9
t_2	$\{a_2, b_1\}$	0110	4
t_3	$\{a_2, b_1, b_2\}$	1110	11
t_4	$\{a_1, a_2, b_2\}$	1011	13

id	contents	gray code	decimal
t_2	$\{a_2, b_1\}$	0110	4
t_1	$\{a_1, b_1, b_2\}$	1101	9
t_3	$\{a_2, b_1, b_2\}$	1110	11
t_4	$\{a_1, a_2, b_2\}$	1011	13

Fig. 7 Sorting the database using gray code ordering

database D we create a signature of I bits by superimposing the signatures of the items that appear in t, i.e., we perform an OR between the signatures. Having transformed database D to a signature database D', we continue by sorting D' according to Gray code ordering. We assume that each signature is a Gray code and we sort them all in ascending order. An example of this procedure is depicted in Figure 7. Since sequential Gray codes have a Hamming distance of one, ordering all signatures according their Gray ordering will bring together transactions that are close in the Hamming distance space. As a result of this ordering, transactions which contain similar sets of items will be placed in the same neighborhood. Thus, by partitioning the Gray-sorted transaction, we have high chances of creating chunks having transactions with common item combinations. This way, fewer generalizations will be needed to anonymize each database chunk.

We have chosen to use gray code ordering as a basis for our partitioning instead of some transaction clustering algorithm for reasons of efficiency. Our basic motivation behind proposing a local recoding anonymization method is to provide a more scalable solution than the AA algorithm. To this end we opted for a computationally cheap solution, like gray code based sorting, over some expensive clustering technique.

One consideration we have to take into account when partitioning a dataset with the aforementioned ordering is the average record size in each chunk. One side product of the Gray-ordering is that transactions of the same size will be positioned close to each other. If we naively split the ordered database in chunks having equal numbers of transactions, then the number of m-sized combinations that appear in each chunk might vary significantly if we have records of various sizes. Having a skewed distribution of the m-sized combinations in the database chunks greatly affects the quality of the solution and the memory requirements. As both the size of the count-tree and of the solution space are directly affected by the number of m-sized combinations that appear in the data, we should create chunks with approximately the same amount of m-sized combinations and not the same amount of transactions. To estimate the total number of combinations an initial scanning of the database has to be performed. Note that the computational overhead is small, since the number of m-sized combinations C of each transaction needs only to be calculated once for each different transaction size s. Then, we simply calculate the product of C and the number of transactions with length s. This calculation introduced only an insignificant overhead in our experiments, which could further be reduced with the help of sampling.

5.3.2 Flexible borders

LRA even with Gray ordering has a weakness; if an item of one class appears very few times in a database chunk, then it will cause all its siblings to be generalized even if they pose no privacy threat. A way to tackle this problem is to allow some *flexibility* in the partitioning of the database. Assume that we have a database and we want to process it in 4 chunks. A straightforward approach would be to split it in four chunks with an equal number of m-sized combinations. When we perform the split, some items might end up at the wrong chunk, i.e., a single appearance of item i, which is very common in the first chunk, might end up at the second chunk. This has a significant impact on the quality of the data. The appearance of a unique item and subsequently unique item combinations, might require generalizing many other sibling items in order to guarantee the k^m-anonymity. To minimize the effect of having items in the wrong chunk, we allow the algorithm to create chunks that deviate up to f transactions from the initial partitioning. The algorithm chooses where to split the database by selecting the pair or successive transactions t_h, t_{h+1} that have the maximum Hamming distance among all other pairs of successive transactions in the region of $[t_{eq-f}, t_{eq+f})$, where t_{eq} is the transaction where the equal size chunk t_{eq} would end.

Algorithm 5 Local Recoding Anonymization

```
   LRA(D, I, k, m)
1: GrayPartitioning($D$,$D_1, \ldots, D_n$)
2: for all $D_i$ in $D_1, \ldots, D_n$ do
3:     $D_{i_{pub}}$:=$AA(D_i, \mathcal{I}, k, m)$
4:     insert $D_{i_{pub}}$ to $D_{pub}$
5: return $D_{pub}$
```

Algorithm 6 Gray ordering & partitioning

```
   ***GrayPartitioning***($D$, $D_1, \ldots, D_n$)
1: $D' = \oslash$                                    ▷ empty signature database
2: **for all** $t$ in $D$ **do**                     ▷ for each record of D
3:     create signature $s_t$ of $t$
4:     insert $s_t$ to $D'$
5: sort $D'$ by Gray code ordering
6: **for all** $D_i$ in $D_1, \ldots, D_n$ **do**
7:     Find the pair ($t_h$,$t_{h+1}$) in $[t_{i\times\frac{|D|}{n}-f}, t_{i\times\frac{|D|}{n}+f})$ of maximum Hamming distance
8:     from unassigned records of $D'$ assign records $t \leq t_h$ to $D_i$
```

The pseudocode for the LRA algorithm is presented in Algorithm 5 and for the Gray code partitioning in Algorithm 6. The main anonymization algorithm simply invokes the *GrayPartitioning* procedure to create the desired database chunks $D_1, \ldots, D_n$, and then anonymizes each chunk using AA. The *GrayPartitioning* procedure first creates the transaction signatures, by setting the bits at the positions corresponding to the items included in each transaction, then sorts the signatures, and finally creates the database chunks, following the flexibility heuristic.

5.3.3 Parallelization

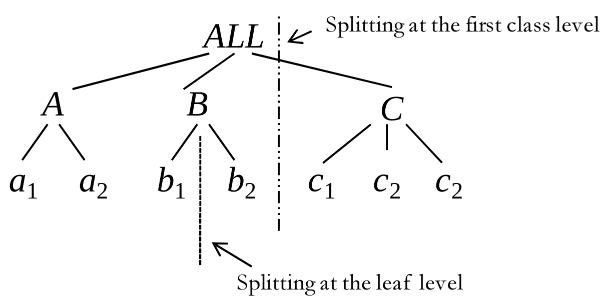
An additional benefit of partitioning the database into chunks and solving the anonymization problem for each of them is that the anonymization procedure can be parallelized. The parallelization of the LRA algorithm is trivial; as the local problems at each chunk are independent, they can directly be parallelized. Thus the total execution time reduces to that of partitioning the dataset plus solving the most expensive sub-problem.

5.4 Anonymization by vertical partitioning

The generalization model as described in Section 3.1 groups together different items that are semantically related, i.e., they belong to the same class. In Section 5.3 we saw that a crucial point in splitting the database is to group related points (and combinations of points) together. The Vertical Partitioning Anonymization algorithm (VPA), instead of splitting the database horizontally, i.e., retaining the original transactions but processing only some of them, it splits the database vertically; all transactions are processed, but only selected items are taken into account at each step. The idea is that if we process only a subset of $\mathcal{I}$, the size of the count tree will be smaller, but at the same time all combinations of these items will be taken into account. There are two important issues in this approach: a) how to select the items that will be processed at each step and b) what to do with the combinations that are composed by items processed at different steps.

The partitioning of the domain $\mathcal{I}$ should group together items that belong to the same class, and classes that belong to the same superclass. Since the generalization model relies on replacing the members of a class with a generalized representative value, the effectiveness of the generalization depends on how many members of a class will appear on the same database part. Independently of the original item and class identifiers, it is possible to sort $\mathcal{I}$ in such a way that each class will contain a contiguous region of items in the order of $\mathcal{I}$ as depicted in Figure 8. For reasons of simplicity we assume that $\mathcal{I}$ is sorted this way, so that we can easily split the domain in the desired number of regions. The partitioning of the domain can take place at different hierarchy levels. If we want to split the database in 3 parts, we can simply split the items that appear in the leaf level of the generalization hierarchy in three equal sized groups. Alternatively, we can perform a partitioning of the items that splits the classes of level 1 in three groups with approximately the same number of items, or the classes of level 2 etc. Creating a partitioning that splits items according to their membership to high level classes has a positive impact on the quality of the data. Since generalization will group together data from the same classes, there is a significant benefit in having all the items of a class in the same chunk. Splitting the items in the leaf level does not guarantee that all classes will have all their items in the same chunk, especially the high level ones. For example, partitioning the items according to their level 1 classes, guarantees that all level 1 classes will have their items in the same chunk. The same holds for any other level. The tradeoff for partitioning $\mathcal{I}$ in a high level is that the algorithm has less control over how many items will end up in each chunk. Different classes might have different sizes and in high levels there might not exist enough classes to uniformly distribute them to different chunks, as in Figure 8. The memory requirements depend on the maximum memory required by the different algorithm steps. If too many items and combinations are examined in one step, the algorithm's memory requirements might be significantly greater compared to a more uniform partitioning.

The pseudo-code for the VPA algorithm is depicted in Algorithm 7. Processing at each step only a part $\mathcal{I}_i$ of $\mathcal{I}$ ensures that all combinations of items from only $\mathcal{I}_i$ are taken into account in the anonymization. The same holds for all parts of $\mathcal{I}$, $\mathcal{I}_1, \ldots, \mathcal{I}_n$. However, combinations that contain items from multiple parts of $\mathcal{I}$ have not been examined at any step (e.g., combination $\{a_1, b_1\}$, assuming $a_1 \in \mathcal{I}_1$ and $b_1 \in \mathcal{I}_2$). To deal with



Splitting at the first class level
ALL
A
B
C
a_1
a_2
b_1
b_2
c_1
c_2
c_2
Splitting at the leaf level

Fig. 8 Splitting the domain at different hierarchy levels

Algorithm 7 Vertical Partitioning Anonymization

```
    VPA(D, I, k, m)
1: partition $\mathcal{I}$ to $\mathcal{I}_1, \ldots, \mathcal{I}_n$
2: for all $\mathcal{I}_i$ in $\mathcal{I}_1, \ldots, \mathcal{I}_n$ do
3:     $D_i$ = project $D$ to $\mathcal{I}_i$
4:     $HC_i$=$AA(D_i, \mathcal{I}_i, k, m)$        ▷ keep only hierarchy cuts
5:     insert $HC_i$ to $HC_{temp}$          ▷ store all temporary $HC$
6: $AA(D, \mathcal{I}, k, m)$                ▷ anonymize based on $HC_{temp}$
```

these combinations we perform a final step where all items of $\mathcal{I}$ are taken into account (i.e., as in the original AA algorithm). To this end, we rescan the database and anonymize the whole data. However, instead of examining all m-sized combinations from $\mathcal{I}$, we consider all generalizations HC_{temp} that have been done at any previous step as *committed*. This means that the anonymization search space is significantly smaller than directly applying AA to D (i.e., without considering HC_{temp}). For example if items a_1 and a_2 have been generalized to item A in HC_{temp}, when we rescan the database we replace any appearance of a_1 or a_2 with A and we trace in the count tree only the appearances on A. The algorithm has an increased I/O cost compared to the AA and LRA algorithms since it must scan the data 3 times and write them once. VPA performs an initial scan to partition D into chunks according to $\mathcal{I}_1, \ldots, \mathcal{I}_n$ and store them to temporary files; it then reads the partitions one by one for anonymization. Finally, the original database in scanned again for the final anonymization. As the dominant factor of the anonymization process is the computational cost, the increased I/O cost of VPA compared to LRA and AA is insignificant compared to the benefit in computational savings, as we will show in Section 6.

Despite the partitioning of the anonymization processes in several steps, VPA performs global recoding, i.e., all the appearances of the items of a class are generalized and not only a part of them. A significant advantage of VPA is that we have explicit control in the partitioning of $\mathcal{I}$, thus better control in the number of item combinations in each partition, which is the dominant cost factor of the anonymization process.

5.4.1 Parallelization

The VPA algorithm, like LRA, can be easily parallelized, as the processing of each chunk can be performed independently. Since there is an essential final pass over the whole database, the cost is determined by the two database scans (one in the beginning for the data partitioning and the final pass in the end) plus the cost of anonymizing the most expensive partition.

6 Experiments

We evaluated experimentally the proposed anonymization techniques, i.e., the OA, DA, AA, LRA and VPA algorithms, by applying them on data stemming from real world applications. The implementation was done in C++ and the experiments were performed on a core-2 duo 2.8GHz CPU server, with 2Gb memory, running Linux.

OA, DA, and AA use the count-tree to evaluate their candidate generalization, thus they avoid scanning the actual database for this purpose. To construct the tree, OA and DA scan the database once, and AA scans it m times (as one tree is constructed for each set cardinality). The tree is kept in main memory at all times. LRA sorts the database and then scans it once for anonymizing each partition, whereas VPA scans it three times as described in Section 5.3. We assume that the local partitions of LRA and VPA fit in memory; thus, although AA is used as a module by these methods to perform local anonymization and AA may have to scan the partition more than once, there is no extra I/O cost. In the following, we detail the experimental setup, the performance factors we trace, and the parameters we vary.

6.1 Experimental Setup

6.1.1 Evaluation metrics

We evaluate our algorithms with respect to three performance factors: a) total execution time, b) memory requirements and c) information loss. We do not measure the I/O cost explicitly, as all methods are CPU-bound (due to the high complexity of checking all m-combinations of items in the worst case). The memory requirements are dominated by the count-tree so we report the memory usage in terms of tree nodes generated. Finally, we measure the information loss using the NCP measure we introduced in Section 3.3.

6.1.2 Evaluation parameters

We investigate how our algorithms behave in terms of several parameters: (i) the domain size $|\mathcal{I}|$, (ii) the database size $|D|$ in terms of number of transactions, (iii) parameters m and k of the k^m-anonymity model, and (iv) the height h of the hierarchy tree, assuming a balanced tree with $|\mathcal{I}|$ leaves.

6.1.3 Datasets

To have a realistic setting for our experimental evaluation, we used three real word-datasets introduced in [32]: *BMS-POS*, *BMS-WebView-1* and *BMS-WebView-2*. Dataset *BMS-POS* is a transaction log from several years of sales of an electronics retailer. Each record represents the products bought by a customer in a single transaction. The *BMS-WebView-1* and *BMS-WebView-2* datasets contain clickstream data from two e-commerce web sites, collected over a period of several months. The characteristics of each dataset are detailed in Figure 9. *All experiments, except those of Fig. 10-12 were run on the original datasets.* We found that the OA and the DA algorithms cannot deal with datasets that have very large domains (DA runs out of memory because it performs no pruning to the combinations of items inserted to the count-tree, and OA does not provide a solution within reasonable response time, i.e., several hours). Still, to be able to evaluate how our heuristics compare to the optimal algorithm in terms of information loss we created several smaller datasets with data originating from *BMS-WebView-2* in the following way. We took the first 2K,5K,10K and 15K records and created four new datasets. Moreover, we had to limit the items domain to only 40 distinct items, so that we could have some results from the OA algorithm which scales the worst compared to the other methods in the domain size. To investigate how OA scales when the domain grows we created two more datasets with 10K records from *BMS-WebView-2* with a domain size of 50 and 60 (for larger domains the algorithm did not respond in a reasonable time). We reduced the domain by performing a modulo on the items *id*s and sequentially removing the duplicates.

Unfortunately, we did not have any real hierarchies for the data, so we constructed some artificial ones. We created those by choosing a fanout for each node in the hierarchy tree, that is a default number of values that are generalized in a direct generalization. For example if we decide on a fanout of n, then each generalization from one level to another generalizes n items. If the size of the domain is not divided by n, some smaller generalization classes can be created. We used an average fanout of 5 for the original datasets and a average fanout of of 4 for the smaller datasets.

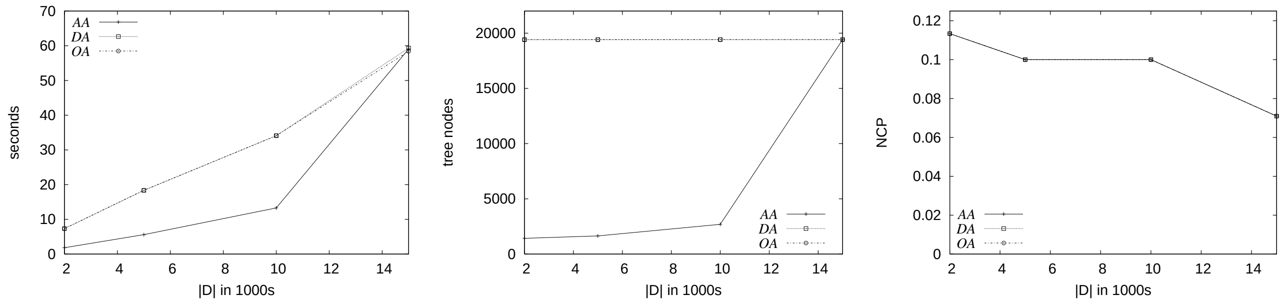
dataset	$\|D\|$	$\|\mathcal{I}\|$	max $\|t\|$	avg $\|t\|$
BMS-POS	515,597	1,657	164	6.5
BMS-WebView-1	59,602	497	267	2.5
BMS-WebView-2	77,512	3,340	161	5.0

Fig. 9 Characteristics of the datasets ($|t|$ stands for transaction size)

6.2 Experimental Results

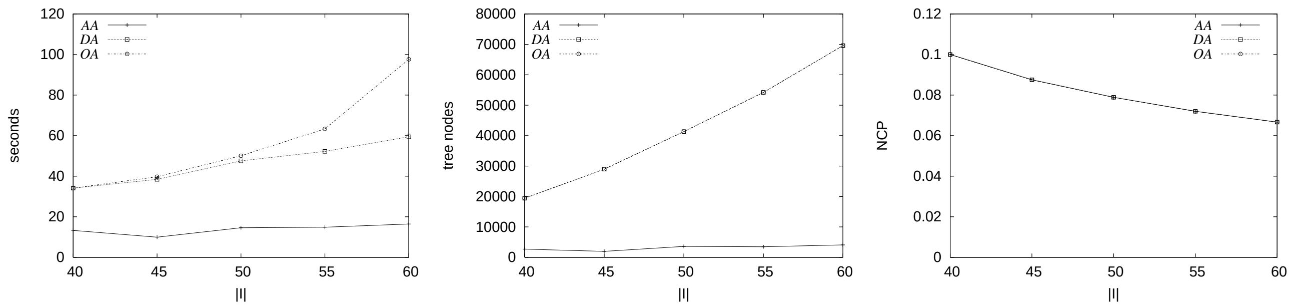
Figures 10a and 10b show how the computational cost and memory requirements of the algorithms scale with the increase of the database size $|D|$, after fixing the remaining parameters to $|\mathcal{I}| = 40$, $k = 100$, $m = 3$, and $h = 4$. As described above, we used prefixes of *BMS-WebView-2* for this purpose. The figure shows that OA and DA have identical performance, which increases linearly to the database size. This is due to the fact that the performance of these methods rely on the size of the count-tree which is not very sensitive to $|D|$, as the distinct number of m-itemset combinations does not increase significantly. On the other hand, AA is initially much faster compared to the other methods and converges to their cost as $|D|$ increases. This is expected, because as $|D|$ increases less i-length itemsets for $i < m$ become infrequent. As a result, AA is not able to significantly prune the space of m-itemsets to be included in the count-tree at its last iteration, due to generalizations performed at the previous loops (where $i < m$). This is also evident from the memory requirements of AA (i.e., the size of the count-tree at the m-th iteration) which converge to the memory requirements of the other two algorithms. Nevertheless, as we will see later, AA does not have this problem for realistic item domain sizes ($|\mathcal{I}|$). Figure 10c shows the information loss incurred by the three methods in this experiment. Note that all of them achieve the same (optimal) loss, which indicates the ability of DA and AA in finding the optimal solution. This behavior is a result of the limited domain size, which leads to a relatively small solution space. In this space the AA and DA manage to find the same, optimal cut.

Figure 11 shows the effect of the domain size $|\mathcal{I}|$ to the performance of the three methods, after fixing $|D| = 10$K, $k = 100$, $m = 3$, and $h = 4$ (again a prefix of *BMS-WebView-2* is used). The results show that the costs of DA and AA increase linearly with the item domain size, however, OA has exponential computational cost with



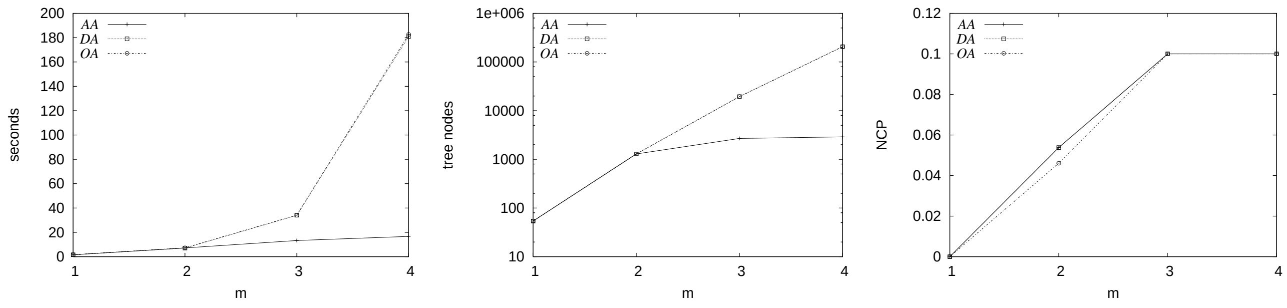
(a) time vs. $|D|$ (b) memory vs. $|D|$ (c) information loss vs. $|D|$

Fig. 10 Effect of database size on the performance of the algorithms



(a) time vs. $|\mathcal{I}|$ (b) memory vs. $|\mathcal{I}|$ (c) information loss vs. $|\mathcal{I}|$

Fig. 11 Effect of domain size on the performance of the algorithms



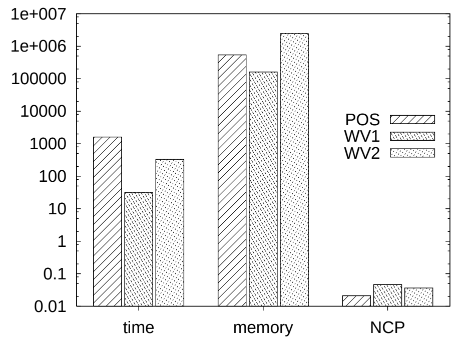
(a) time vs. m (b) memory vs. m (c) information loss vs. m

Fig. 12 Effect of m on the performance of the algorithms

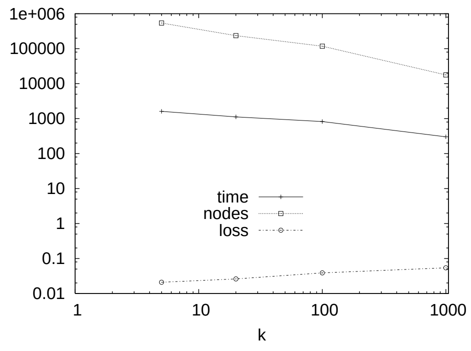
respect to $|\mathcal{I}|$, as expected by our analytical results. The memory requirements of OA and DA increase super-linearly with $|\mathcal{I}|$, while AA achieves better scalability, due to its ability to prune items at early iterations. This does not compromise the information loss of the algorithm, which is the same as that of the optimal solution by OA.

In the next experiment, we change the value of m, fix the values of other experimental parameters ($|D| =$ 10K, $|\mathcal{I}| = 40$, $k = 100$, $h = 4$), and compare the three algorithms on the three performance factors. As shown in Figure 12, the CPU-time of OA and DA does not scale well with m (exponential), while the cost of AA increases linearly. This is due to the exponential increase at the size of the count-tree used by these two algorithms. Again, AA achieves better scalability, due to the early generalization of a large number of items. The information loss of DA and AA is very close to that of the optimal solution.

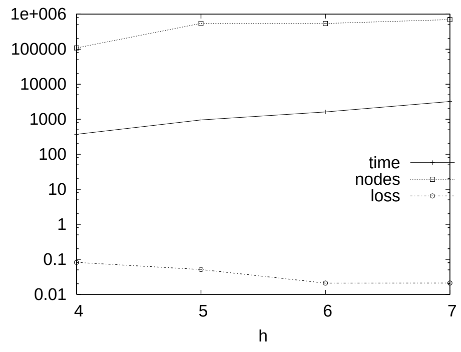
In the next set of experiments (depicted in Figure 13), we measure the computational cost (in msec), memory requirements (in number of nodes in the count-tree), and information loss of the AA algorithm for large, realistic problems, where OA and DA cannot run. We used the exact three datasets as described in Figure 9, abbreviated as POS, WV1, and WV2, respec-



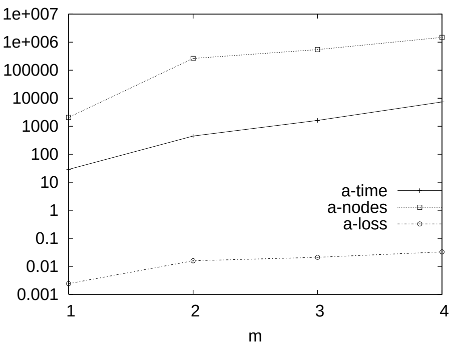
(a) AA on three real datasets



(b) effect of k (POS)



(c) effect of h (POS)



(d) effect of m (POS)

Fig. 13 Apriori algorithm on the original datasets

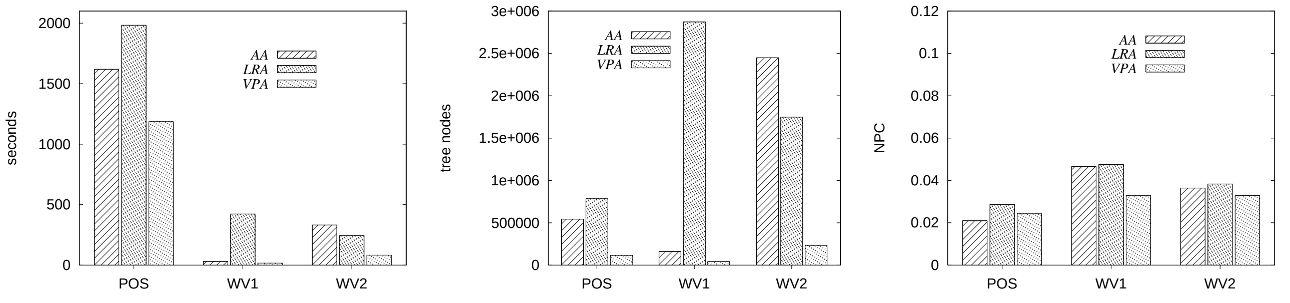
tively. First, we show the performance of AA on all three datasets by setting m=3, k=5, and the fanout of each node in the generalization hierarchies of all domains to 5 (this results in $h = 6$, $h = 5$, and $h = 7$, for POS, WV1, and WV2, respectively). Figure 13a shows that AA runs in acceptable time, generating a manageable count-tree, and producing a solution of low information loss (maximum 3%). Figure 13b shows the performance of AA on the POS dataset, after varying k and keeping the values of m and h fixed. The plot shows that both the computational cost and the memory requirements are insensitive to k. Figure 13c, fixes m and k and varies h. Note that for smaller values of h AA is faster but produces worse solutions. For $h = 4$, in specific, AA fails to find a non-trivial generalization of the dataset (we noted that this was the only experimental instance where actually AA performed badly). For all other values the quality of the solution is very good (information loss close to 1%). Finally, Figure 13d shows how the performance is affected by varying m while fixing k and h. Time increases as m grows and the solution found becomes slightly worse in terms of information loss (but within acceptable limits). This is expected, as m increases the maximum length of itemsets to be checked and the size of the tree, accordingly.

In summary, AA is a powerful heuristic for solving k^m-anonymity problems of realistic sizes, where the application of an optimal solution is prohibitively expensive.

6.2.1 The LRA and VPA algorithms

The performance of the LRA and VPA algorithms is evaluated in Figures 14-19. Again we measure total execution time, maximum memory required in terms of tree nodes and the NPC information loss as defined in Section 3.3. Unless explicitly stated otherwise, we perform our experiments on the *BMS-POS* dataset, with $k = 5, m = 3, flexibility = 10$ (for LRA), $partition\ level =$ 1 (for VPA; 0 is the leaf level) and we split the database in 3 parts for both the LRA and VPA algorithms.

Figure 14 shows how the algorithms behave on all the three datasets. We also plot the performance of AA as a point of reference. A surprising result at a first glance is the poor performance of the LRA method. While the information loss is only slightly increased w.r.t. to AA, the LRA has significantly greater memory requirements and execution time, especially for WV1. The explanation lies in the skewed distribution of record sizes in WV1. This dataset has average record length of 2.5, but contains some very large records, with more



(a) time (b) memory (c) information loss

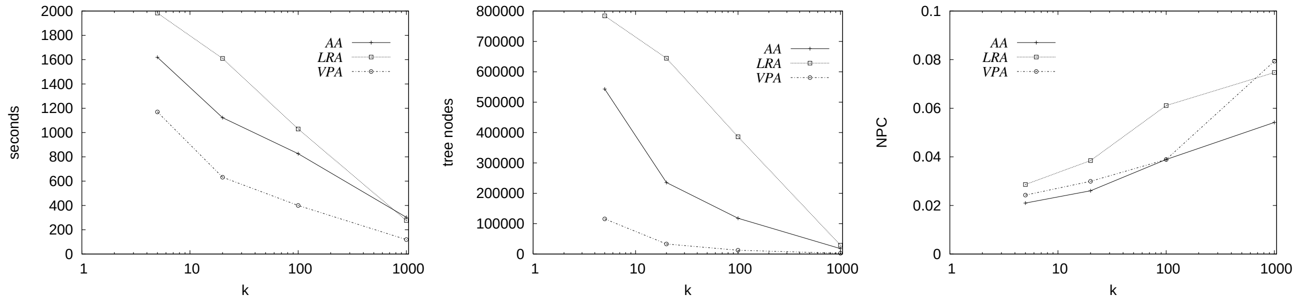
Fig. 14 Performance on the three datasets

than 200 items each. The Gray code ordering brings together records that not only have small Hamming distance, but they also have similar sizes. LRA partitions the database w.r.t. to the number of m-sized combinations that lie in each chunk, thus we end up with chunks that have few large records and others that have numerous small records. Large records support all combinations between their items, i.e., they completely support a subspace of the space of combinations of the items of $\mathcal{I}$. As a consequence the chunks with few large records usually have few privacy threats and few generalizations are performed. The fewer generalizations greatly reduce the pruning in the the count-tree, thus the memory requirements and the time to construct the larger count tree is greatly increased for these chunks. As it can be seen in Figure 18, if we increase the number of chunks we can limit the memory requirements, but still LRA cannot beat VPA.

The most efficient method, as indicated by Figure 14, is VPA. It requires significantly less memory and execution time than the other algorithms and incurs an information loss similar to the information loss of AA. The superiority of VPA lies in the fact that we have better control over the partitioning of the anonymization space, which is affected mainly by the number of item combinations considered at the anonymization. This number is significantly smaller at the partitions of VPA, which are well-balanced according to this factor, compared to the partitions of LRA and of course the full itemset space examined by AA. Moreover, in the final step of VPA, when the whole dataset is anonymized, the solution space has already been substantially pruned. In Figure 14c we can see that VPA manages to have marginally better information loss that AA. As they are both heuristics, there is no guarantee on their relative information loss. The two methods perform different generalizations at the initial steps, which sometimes turn out to be better in the case of VPA. This result does not reflect the general case; our complete experimental view shows that AA is in generally (slightly) better than VPA in terms of information loss.

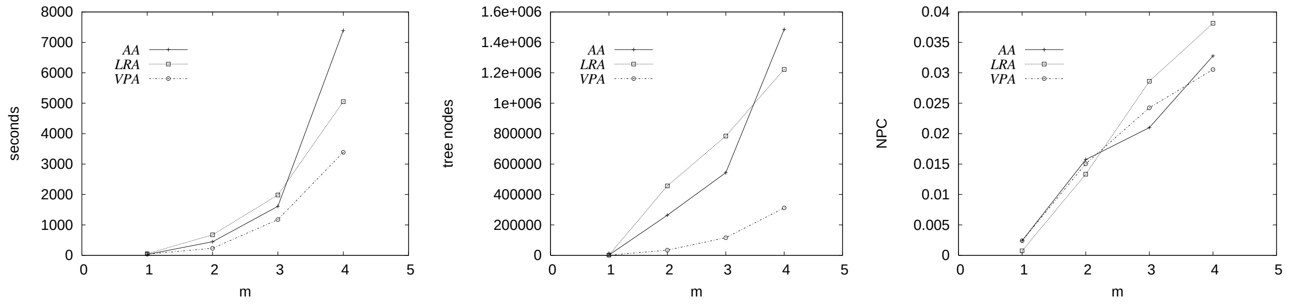
Figures 15 and 16 depict the effect that k and m have in the performance of the algorithms (using the *BMS-POS* dataset). First we fix all other parameters and we vary $k = 5, 20, 100, 1000$. The VPA algorithm is significantly faster and requires less memory than the other two, offering an information loss comparable to AA's for small k. For a very large k, all algorithms are fast and have small memory requirements, since the generalizations that take place at the first level of AA (which is used by both LRA and VPA) significantly reduce the domain. At the same time, we can see that the information loss introduced by the LRA and VPA becomes significantly larger for $k = 1000$, since it is harder for them to create combinations with support k in the partition of the database that is at their disposal at each step. To examine the effect of m, we fix all other parameters and vary $m = 1, 2, 3, 4$. As m grows and more item combinations must be examined, VPA and LRA scale much better than AA, with VPA consistently being much faster than the other methods. Both LRA and VPA demonstrate a behavior similar to AA in terms of information loss. Figure 17 shows the behavior of the algorithms w.r.t. to different generalization hierarchy heights. A high level signifies a more refined hierarchy and less information loss. On the other hand, the increase of h increases the solution space, meaning that the algorithms may have to spend more effort to reach a solution. Note that VPA has significantly smaller memory requirements than the other two methods. LRA and AA have similar performance, with LRA being better when the partitioning manages to cluster items from the same classes well, and worse when it does not.

In Figure 18 we investigate the effect of factors unique to the LRA and VPA algorithms. Figures 18a and 18b show the effect of database partitioning. In the horizontal axis we vary the number of partitions in the data. For the LRA algorithm this reflects database chunks



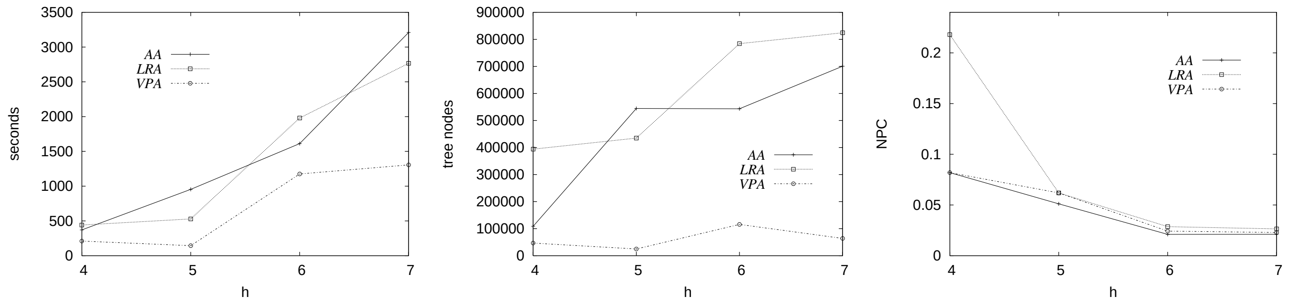
(a) time vs. k (b) memory vs. k (c) information loss vs. k

Fig. 15 Effect of k on the performance of the algorithms



(a) time vs. m (b) memory vs. m (c) information loss vs. m

Fig. 16 Effect of m on the performance of the algorithms

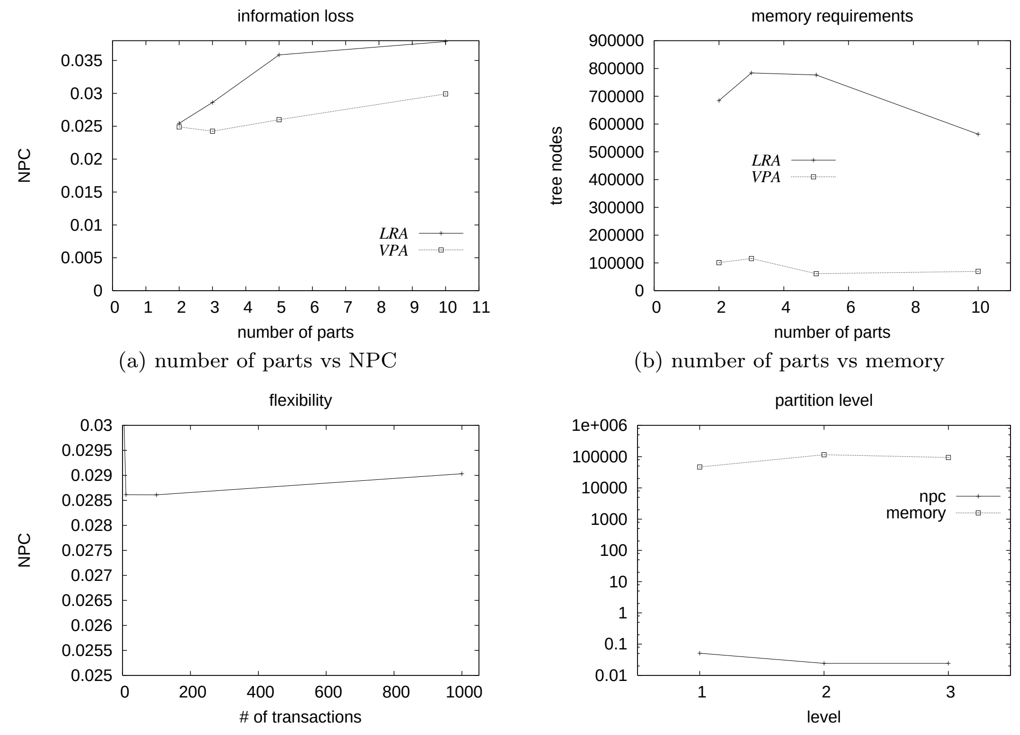


(a) time vs. h (b) memory vs. h (c) information loss vs. h

Fig. 17 Effect of hierarchy level h on the performance of the algorithms

and for the VPA algorithm the domain parts. As expected, when the number of partitions grow, the memory requirements and the quality of the data fall. This is not a strictly monotonous procedure; as the database chunks get smaller there is a chance that fewer rare items might appear, thus the other items of the same class will not be generalized in the first iteration of AA. This effect explains the small increase of memory requirements for the LRA algorithm. In Figures 18c and 18d we see how flexibility affects the LRA algorithm and how the *partitioning level* affects VPA. We observe that these factors do not have a significant effect in the performance of the methods. We omit time for Figure 18d and time and memory for Figure 18c, as they remain approximately stable.

In the set of experiments depicted in Figure 19 we measure the total time and the total memory needed by LRA and VPA if they are used to parallelize the anonymization of the data. LRA does not benefit much from parallelization due to the fact that there usually exists an expensive partition to be serially processed that dominates the overall cost. As explained, with LRA we may end up with a chunk of few but very long transactions. This chunk is very hard to anonymize as there is a huge number of item combinations that do not violate k^m-anonymity. For this chunk, the AA module of LRA



(a) number of parts vs NPC
(b) number of parts vs memory
(c) effect of flexibility
(d) effect of partitioning level

Fig. 18 Factors affecting the LRA and VPA algorithms

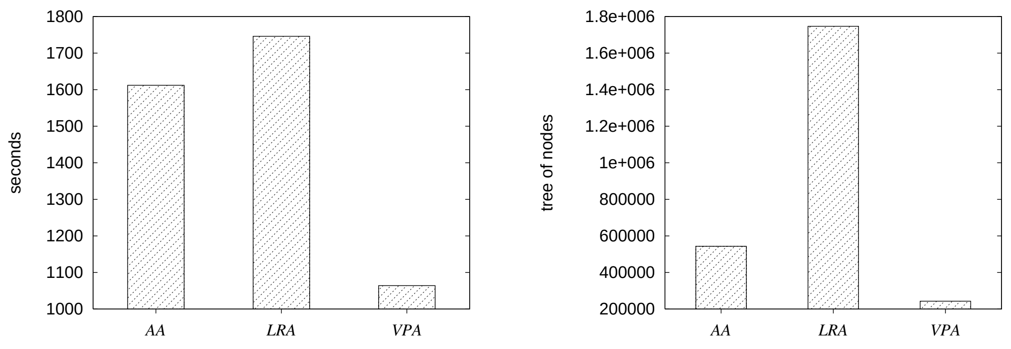


Fig. 19 Time and memory requirements in parallel execution

cannot perform many generalizations at its first iterations, therefore a huge count tree is constructed. On the other hand, the partitions generated by VPA are balanced with respect to the item combinations and VPA benefits the most by parallelization. In summary, VPA is the most robust method for k^m-anonymity, which can accelerate AA to a high degree, especially with the use of parallelization, while it does not compromise the quality of the result.

Finally, in Figures 20-22 we evaluate the best algorithms we have according to the previous experiments, AA and VPA, in terms of the ML^2 and dML^2 information loss measures, which we introduced in Section 3.3. To measure ML^2 and dML^2 we mined the original and the anonymized datasets for frequent itemsets in all generalization levels, using support thresholds 1%, 2% and 3%. The results presented in Figure 20 are based on the average values of all three thresholds for each dataset. Both ML^2 and dML^2 follow the NPC results depicted in Figure 14. ML^2 (measured in percentage points) shows the fraction of frequent itemsets that are not detected in the appropriate level of the generalization hierarchy in the anonymized dataset. dML^2 shows the difference between the actual level the itemsets and the level they were detected when using the anonymized dataset. Note, that in dML^2 we take into account only

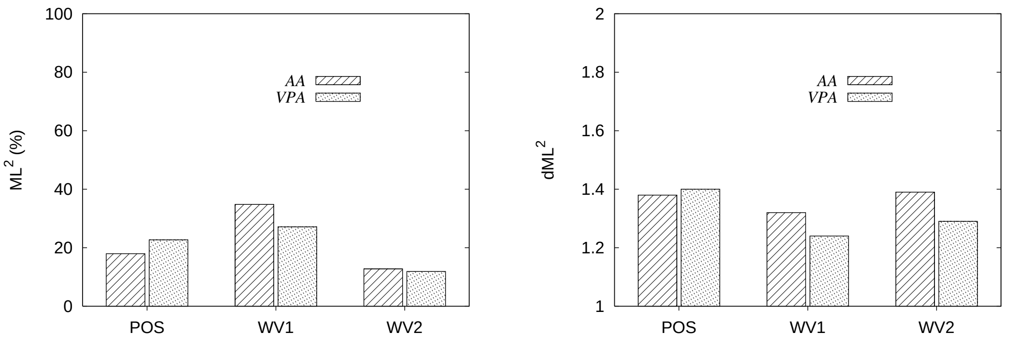


Fig. 20 The ML^2 and dML^2 information loss measures

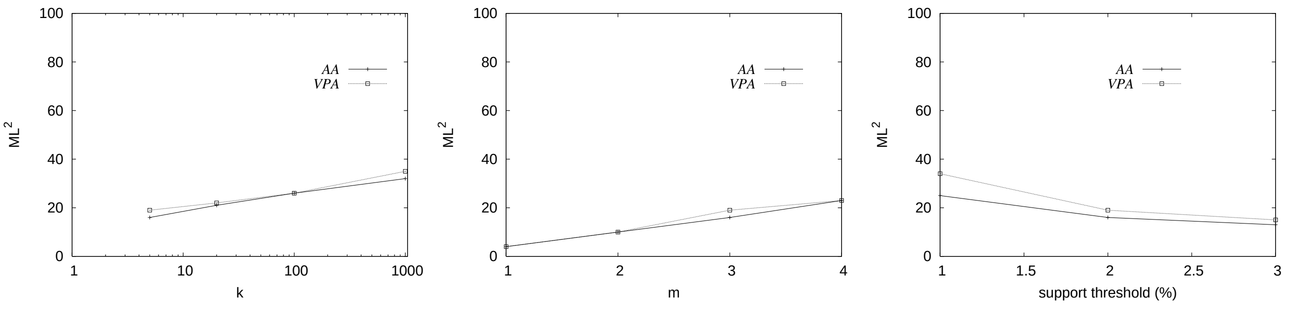


Fig. 21 Behavior of ML^2 with respect to k,m and the frequent itemset support threshold

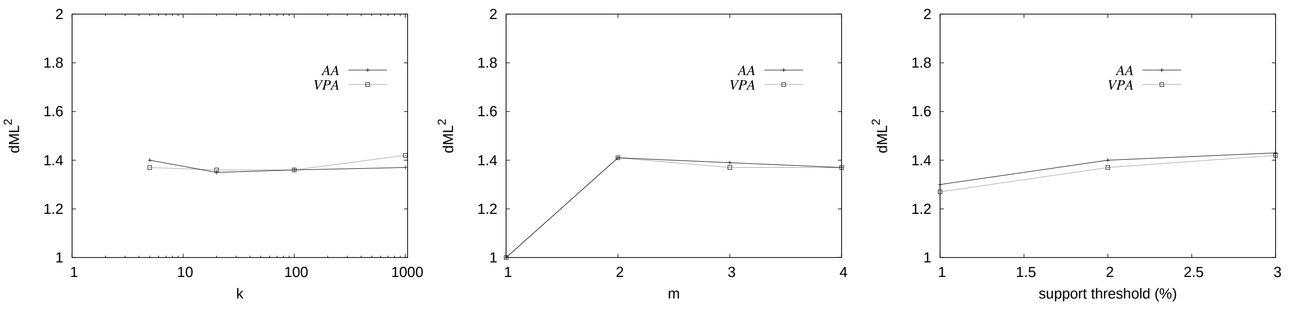


Fig. 22 Behavior of ML^2 with respect to k,m and the frequent itemset support threshold

the frequent itemsets that were not appropriately detected, thus the theoretical minimum is 1. The anonymization was performed by using the default parameters for both AA and VPA. The results show that the percentage of frequent itemsets which are detected only at a more generalized level is low in general. In addition, the level at which they are detected is not much higher than their actual level. This implies that the anonymized datasets by our methodology do not have a negative effect on the results of frequent itemset mining methods, since frequent itemsets usually satisfy k^m anonymity and they are not generalized; even in the cases where they are generalized, the information loss is not very high as they are found only 1 or 2 levels above in the generalization hierarchy. In Figure 21 we explore in more depth the effect of k, m and the support threshold we used to retrieve the frequent itemsets. We fixed all parameters in the default values (we used 2% as the default support threshold) and varied each time only one parameter. As is the case of NPC, the ML^2 grows linearly with k and m. On the other hand, as the support threshold grows, less frequent itemsets are detected in the low levels of the generalization hierarchy. This means that the impact of the anonymization procedure, which hides data on these levels, is reduced, and this is reflected in the decrease of ML^2. In Figure 22 we can observe how dML^2 is affected by the same parameters. The general observation is that apart from the case of $m = 1$, where the anonymization has significantly less impact on the dataset, dML^2 does not demonstrate a great volatility. Thus, even if the number of frequent itemsets that are not appropriately detected in some generalization level changes, these frequent itemsets are found in a fixed higher generalization level.

6.2.2 Comparison with Partition *Algorithm*

In [12] the authors propose a top-down local recoding algorithm, named *Partition*, for preserving the privacy in the publication of data that contain set-values. Although the focus on the paper is on full k-anonymity, they present an experimental comparison with AA. The comparison investigates the difference in the information loss between the k^m anonymity achieved by the AA and the k-anonymity achieved by the Partition algorithm for different values of k and m. Their results show that for small m the information loss of AA is less than that of Partition and that Partition is significantly faster than AA. In Figure 23, we complement the experimental evaluation of our algorithms by showing how the Partition compares to the most efficient of them, i.e., the VPA. We used our implementation for VPA and the implementation of [12] for the Partition algorithm, which we acquired from the authors. Using again the default parameter values, we performed the experiments on *BMS-POS* dataset. The results on Figure 23 support the same results with [12]; k^m anonymity by the VPA algorithm inccurs less information loss for a small m than k-anonymity with Partition and the picture is reversed for a large m. Notice that the average record size for *BMS-POS* is 6.5, thus the point where Partition outperforms VPA ($m = 5$) is quite large. The experiments show that when we fix $m = 3$ and vary k, the VPA retains its advantage. In terms of evaluation time, Partition is a lot faster; it takes only ~ 20 seconds, whereas VPA starts from ~ 40 seconds for $m = 1$ and goes up to a bit less than 5 hours for $m = 6$. This is due to the fact that VPA, being a bottom up algorithm, runs well when few generalizations are needed but it is slow when the top levels of the generalization hierarchy must be reached.[4]

A final point for VPA and LRA is that they have good parallelization properties and they can take advantage of parallel architectures like map/reduce [6]. Partition has parallelization properties but offers less explicit control on the partitioning, since the size of the partitions is not set explicitly but it it follows the generalization hierarchy (in top-down order). In contrast we can directly choose how many records or how many items will fall in each partition of LRA and VPA respectively.

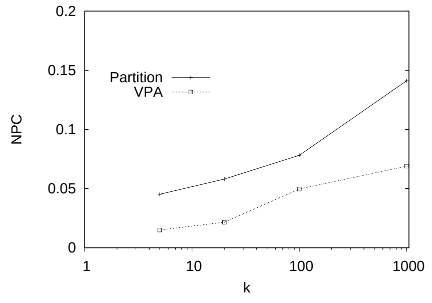
[4] The information loss depicted in Figure 23 is slightly lower for VPA than that reported in the previous experiments. This is due to the fact that we had a to create a different generalization hierarchy for this experiment, so that it would be compatible with the Partition implementation. The previous hierarchy had an *average* fanout of 5, whereas this one has a *constant* fanout of 5.

7 Conclusions

In this paper, we studied the k-anonymization problem of set-valued data. We defined the novel concept of k^m-anonymity for such data and analyzed the space of possible solutions. Based on our analysis, we developed an optimal, but not scalable, algorithm which is not practical for large, realistic databases. In view of this, we developed two greedy heuristic methods, of lower computational cost, which find near-optimal solutions. Our apriori-based anonymization algorithm, in specific, has low memory requirements, making it practical for real problems.

We complemented our work with two partitioning-based approaches aiming at the reduction of the memory consumption and execution cost. The local recoding anonymization algorithm is an intuitive method, which partitions the database horizontally and solves the problem at each part. However, this method does not perform well in practice, because it is hard to create good clusters of transactions due to the extremely high dimensionality of the problem. Although a sophisticated Gray-ordering scheme has been used for partitioning, it is often the case that there exists a partition which is very expensive to anonymize and becomes the bottleneck of the whole process. To overcome these problems, we proposed a vertical partitioning algorithm, which splits the hierarchical classification of the items into a forest of subtrees, and projects the database at each subtree, creating a partition. The partitions are then anonymized and a final pass is applied on the database to handle privacy breaches caused by itemsets with items from different partitions. The vertical partitioning algorithm is shown to be the best method in practice, as it creates balanced partitions that can be processed very fast. We have shown that this method can also greatly benefit from parallelization.

We emphasize that our techniques are also directly applicable for databases, where tuples contain both a set-valued attribute and other sensitive attributes. In this case, k-anonymity with respect to all m-subsets of the domain of the set-valued attribute can help avoiding associating the sensitive value to less than k tuples. In the future, we aim at extending our model to ℓ-



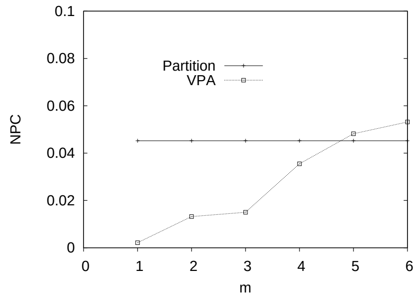


Fig. 23 Comparison of VPA with Partition algorithm

diversity considering sensitive values associated to set-valued quasi-identifiers.

Acknowledgments

We would like to thank the authors of [12] for sharing with us the implementation of the Partition algorithm. This work was supported by grant HKU 715108E from Hong Kong RGC.

References

1. C. C. Aggarwal. On k-anonymity and the curse of dimensionality. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 901–909. VLDB Endowment, 2005.
2. G. Aggarwal, T. Feder, K. Kenthapadi, S. Khuller, R. Panigrahy, D. Thomas, and A. Zhu. Achieving Anonymity via Clustering. In *Proc. of ACM PODS*, pages 153–162, 2006.
3. G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Approximation Algorithms for k-Anonymity. *Journal of Privacy Technology*, (Paper number: 20051120001), 2005.
4. M. Atzori, F. Bonchi, F. Giannotti, and D. Pedreschi. Anonymity Preserving Pattern Discovery. *VLDB Journal*, accepted for publication, 2008.
5. R. J. Bayardo and R. Agrawal. Data Privacy through Optimal k-Anonymization. In *Proc. of ICDE*, pages 217–228, 2005.
6. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. pages 137–150, December 2004.
7. G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis. Fast Data Anonymization with Low Information Loss. In *vldb*, pages 758–769, 2007.
8. G. Ghinita, Y. Tao, and P. Kalnis. On the Anonymization of Sparse High-Dimensional Data. In *Proc. of ICDE*, 2008.
9. J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *vldb*, pages 420–431, 1995.
10. J. Han and Y. Fu. Mining multiple-level association rules in large databases. *IEEE TKDE*, 11(5):798–805, 1999.
11. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. of ACM SIGMOD*, pages 1–12, 2000.
12. Y. He and J. F. Naughton. Anonymization of set-valued data via top-down, local generalization. *PVLDB*, 2(1):934–945, 2009.
13. V. S. Iyengar. Transforming Data to Satisfy Privacy Constraints. In *Proc. of SIGKDD*, pages 279–288, 2002.
14. K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient Full-domain k-Anonymity. In *Proc. of ACM SIGMOD*, pages 49–60, 2005.
15. K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian Multidimensional k-Anonymity. In *Proc. of ICDE*, 2006.
16. N. Li, T. Li, and S. Venkatasubramanian. t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In *Proc. of ICDE*, pages 106–115, 2007.
17. A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-Diversity: Privacy Beyond k-Anonymity. In *Proc. of ICDE*, 2006.
18. A. Meyerson and R. Williams. On the Complexity of Optimal K-anonymity. In *Proc. of ACM PODS*, pages 223–228, 2004.
19. M. Nergiz, C. Clifton, and A. Nergiz. Multirelational k-anonymity. Technical Report CSD TR 08-002.
20. M. Nergiz, C. Clifton, and A. Nergiz. Multirelational k-anonymity. In *Proc. of ICDE*, pages 1417 – 1421, 2007.
21. M. E. Nergiz and C. Clifton. Thoughts on k-anonymization. *Data and Knowledge Engineering*, 63(3):622–645, 2007.
22. H. Park and K. Shim. Approximate algorithms for k-anonymity. In *Proc. of ACM SIGMOD*, pages 67–78, 2007.
23. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C, 2nd Edition.* Cambridge University Press, 1992.
24. P. Samarati. Protecting Respondents' Identities in Microdata Release. *IEEE TKDE*, 13(6):1010–1027, 2001.
25. L. Sweeney. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.
26. M. Terrovitis, N. Mamoulis, and P. Kalnis. Privacy-preserving Anonymization of Set-valued Data. *Proceedings of the VLDB endowment (PVLDB) (former VLDB proceedings)*, 1(1), 2008.
27. V. S. Verykios, A. K. Elmagarmid, E. Bertino, Y. Saygin, and E. Dasseni. Association Rule Hiding. *IEEE TKDE*, 16(4):434–447, 2004.
28. X. Xiao and Y. Tao. Anatomy: Simple and Effective Privacy Preservation. In *Proc. of VLDB*, pages 139–150, 2006.
29. 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.
30. Y. Xu, K. Wang, A. W.-C. Fu, and P. S. Yu. Anonymizing transaction databases for publication. In *Proc. of KDD*, pages 767–775, 2008.
31. Q. Zhang, N. Koudas, D. Srivastava, and T. Yu. Aggregate Query Answering on Anonymized Tables. In *Proc. of ICDE*, pages 116–125, 2007.
32. Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *Proc. of KDD*, pages 401–406, 2001.