Privacy-preserving Anonymization of Set-valued Data

Manolis Terrovitis
Dept. of Computer Science
University of Hong Kong
rrovitis@cs.hku.hk

Nikos Mamoulis
Dept. of Computer Science
University of Hong Kong
nikos@cs.hku.hk

Panos Kalnis
Dept. of Computer Science
National University of Singapore
kalnis@comp.nus.edu.sg

ABSTRACT

In this paper we study the problem of protecting privacy in the publication of set-valued data. Consider a collection of transactional data 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 two greedy heuristics, which scale much better and in most of the cases find a solution close to the optimal. The proposed algorithms are experimentally evaluated using real datasets.

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 [18, 19], 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.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand

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, "skim-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 algorithms in this direction. Our first *optimal anonymization* (OA) algorithm 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 two heuristics 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., "skim-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 the most 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.

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. Section 5 includes an experimental evaluation, and Section 6 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 [18, 19]. Records with identical QI values form an anonymized group. Two techniques to preserve privacy are generalization and suppression [19]. 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→state→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.

[13] 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 $O(k \cdot logk)$. [2] improved this bound to $O(k)$, while [17] further reduced it to $O(\log k)$. Several approaches limit the search space by considering only global recoding. [4] 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* [9] 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→continent→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 [16], 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* [10] partitions the space recursively across the dimension with the widest normalized range of values and supports a limited version of local recoding. [1] 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. [22] 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 mod-

eling 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. [15, 14] 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 [12] 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 anonymizing group (e.g., all persons suffer from the same disease). [21, 23, 5] present various methods to solve the ℓ-diversity problem efficiently. [6] extends [21] 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. [11] 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. We plan to explore variations of the l-diversity for our setting in the future.

Related issues were also studied in the context of data mining. [20] 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 [3], 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. [3] 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.

3. PROBLEM SETTING

Let D be a database containing $|D|$ transactions. Each transaction $t \in D$ is a *set* of items.[1] Formally, t is a non-empty subset of $\mathcal{I} = \{o_1, o_2, \ldots, o_{|\mathcal{I}|}\}$. $\mathcal{I}$ 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:

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 "skim-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., {skim-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.

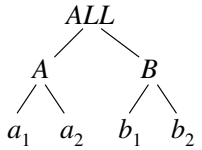


Figure 1: Sample generalization hierarchy

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

[1]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.

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* $\forall o(o \in t) \Leftrightarrow ((o \in gt) \vee (g(o) \in gt))$

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')

Figure 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, the support of $\{a_1, a_2\}$ in D' is the support of $\{A\}$ in D', that is 4.

We opt for a *global recoding* approach [4, 9] which applies the selected generalization rules to *all* transactions in the database. An example of global recoding has already been shown in Figure 2. An alternative *local recoding* generalization [10, 1] 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, however, it explodes the possible space of solutions, making the k^m-anonymity problem even harder. The application of local recoding techniques to solve this problem is a topic of future work.

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.

The trivial generalization $\mathcal{I} \rightarrow ALL$, suppresses the whole database, since all items are replaced by a generalized value

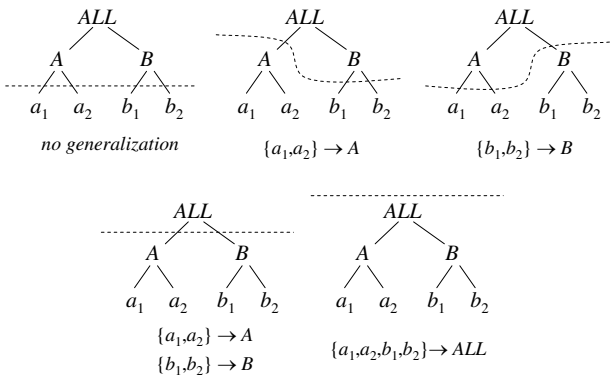


Figure 3: Possible domain generalizations

(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 $\langle A, b_1, b_2 \rangle$.

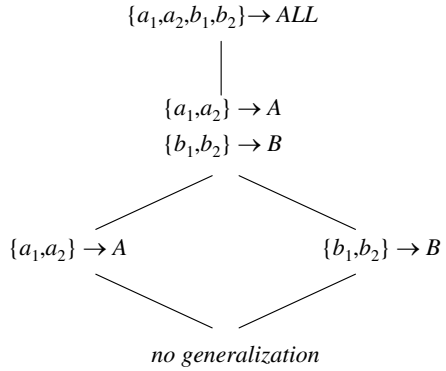


Figure 4: Hierarchy of domain generalizations

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) [8] is suitable when the purpose of the anonymized data is to train a classifier, whereas the *Discernibility Metric* (DM) [4] measures the cardinality of the anonymized groups. More accurate is the *Generalized Loss Metric* [8] and the similar *Normalized*

Certainty Penalty (NCP) [22]. 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 $\frac{2 \cdot 0.5 + 3 \cdot 0.5 + 0 + 0}{11} = \frac{2.5}{11}$.

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 [9]. 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 these 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 [7]. 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_2) 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, $\{a_2, b_1\}$, $\{a_2, B\}$, $\{b_1, A\}$, $\{A, B\}$.

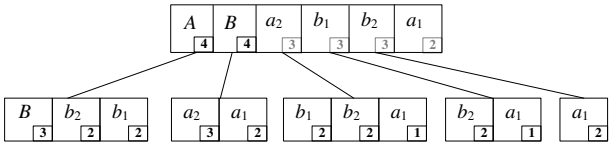


Figure 5: Count-tree for the database of Figure 2

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

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 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.

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	$\{a_1, a_2, b_2, A, B\}$

Figure 6: Expanded Database

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 our first heuristic algorithm, called *direct anonymization* (DA), is to scan the count-tree once for possible privacy breaches and then use the generalized combi-

nations 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 $\{a_1, a_2\} \rightarrow A$ and $\{a_1, a_2, b_1, b_2\} \rightarrow ALL$ 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., $\{a_1, a_2\} \rightarrow A$). 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.

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}

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 $\{a_1, a_2\} \rightarrow A$, 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 $\langle\{a_1, a_2\} \rightarrow A\rangle$.

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 1 item from each trajectory, then 2 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 combination 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. EXPERIMENTS

We evaluated experimentally the proposed anonymization techniques, i.e., the OA, DA and AA algorithms, by applying them on data stemming from real world applications. In the following we detail the experimental setup, the performance factors we trace and the parameters we vary.

All algorithms use the count-tree to evaluate their candidate generalization, thus they avoid scanning the actual database (OA and DA scan the database once and AA scans it m times). The tree is kept in main memory at all times. 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.

5.1 Experimental Setup

5.1.1 Evaluation metrics

We evaluate our algorithms with respect to three performance factors: a) total execution time, b) memory requirements and c) information loss. Since all our three algorithms use the tree to detect privacy breaches and evaluate candidate solutions, the total time is basically CPU time.[2] Moreover, 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.

5.1.2 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, where each non-leaf node has the same degree.

5.1.3 Datasets

To have a realistic setting for our experimental evaluation, we used three real word-datasets introduced in [24]: *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 7. 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 worse 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 *degree* 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 degree 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 a degree 5 for the original datasets and a degree of 4 for the smaller datasets.

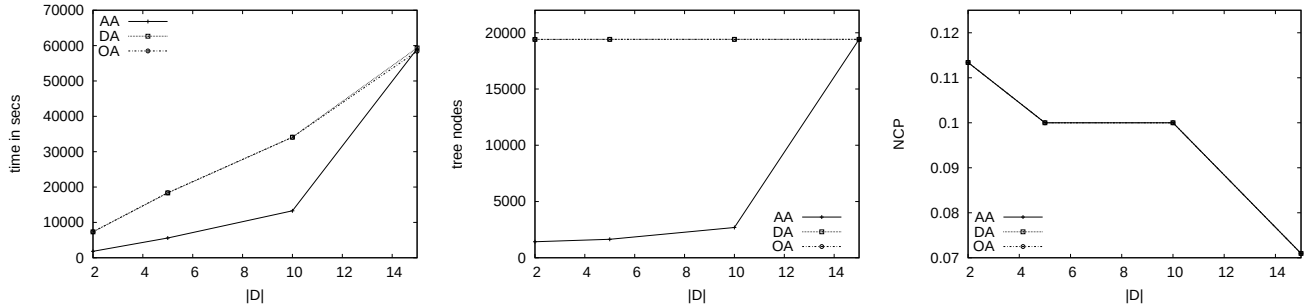
dataset	$\|D\|$	$\|\mathcal{I}\|$	max $\|t\|$	avg $\|t\|$
BMS-POS	515,597	1,657	164	6.5
BMS-WebView-1	59,602	497	267	2.5
BMS-WebView-2	77,512	3,340	161	5.0

Figure 7: Characteristics of the datasets ($|t|$ stands for transaction size)

5.2 Experimental Results

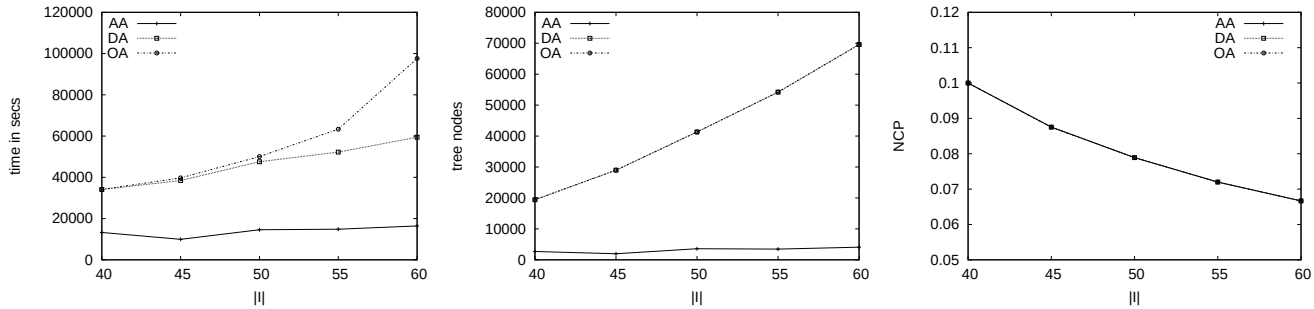
Figures 8a and 8b show how the computational cost and memory requirements of the algorithms scale with the increase of the database size $|D|$, after fixing the remaining parameters to $|\mathcal{I}| = 40$, $k = 100$, $m = 3$, and $h = 4$. The figure shows that OA and DA have identical performance, which 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,

[2]Although AA scans the database m times, instead of 1 time by OA and DA, the cost of this additional I/O is negligible compared to the savings in computation.



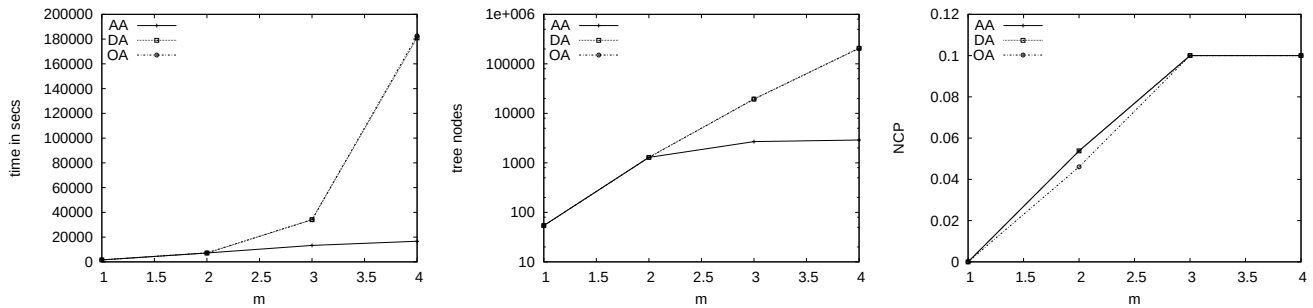
(a) time vs. $|D|$ (b) memory vs. $|D|$ (c) information loss vs. $|D|$

Figure 8: Effect of database size on the performance of the algorithms



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

Figure 9: Effect of domain size on the performance of the algorithms



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

Figure 10: Effect of m on the performance of the algorithms

AA does not have this problem for realistic item domain sizes ($|\mathcal{I}|$). Figure 8c 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 9 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 $h = 4$. The results show that the costs of DA and AA increase linearly with the item domain size, however, OA has exponential computational cost with 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 ($|D| = 10$K, $|\mathcal{I}| = 40$, $k = 100$, $h = 4$), and compare the three algorithms on the three performance factors. As shown in Figure 10, 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 last set of experiments (depicted in Figure 11), 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 POS, WV1 and WV2 datasets as described in Figure 7. First, we show the performance of AA on all three datasets by setting m=3, k=5, and the degree 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 11a shows that AA runs in acceptable time, generating a manageable count-tree, and producing a solution of low information loss (maximum 3%). Figure 11b 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 11c, 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 11d 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. 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, is scalable and has low memory requirements, making it practical for real problems.

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-diversity considering sensitive values associated to set-valued quasi-identifiers. We will also investigate local recoding techniques, which will potentially result in lower information loss by the published data.

Acknowledgments

This work was supported by grant HKU 7149/07E from Hong Kong RGC.

7. REFERENCES

[1] 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.

[2] 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.

[3] M. Atzori, F. Bonchi, F. Giannotti, and D. Pedreschi. Anonymity Preserving Pattern Discovery. *VLDB Journal*, accepted for publication, 2008.

[4] R. J. Bayardo and R. Agrawal. Data Privacy through Optimal k-Anonymization. In *Proc. of ICDE*, pages 217–228, 2005.

[5] G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis. Fast Data Anonymization with Low Information Loss. In *vldb*, pages 758–769, 2007.

[6] G. Ghinita, Y. Tao, and P. Kalnis. On the Anonymization of Sparse High-Dimensional Data. In *Proc. of ICDE*, 2008.

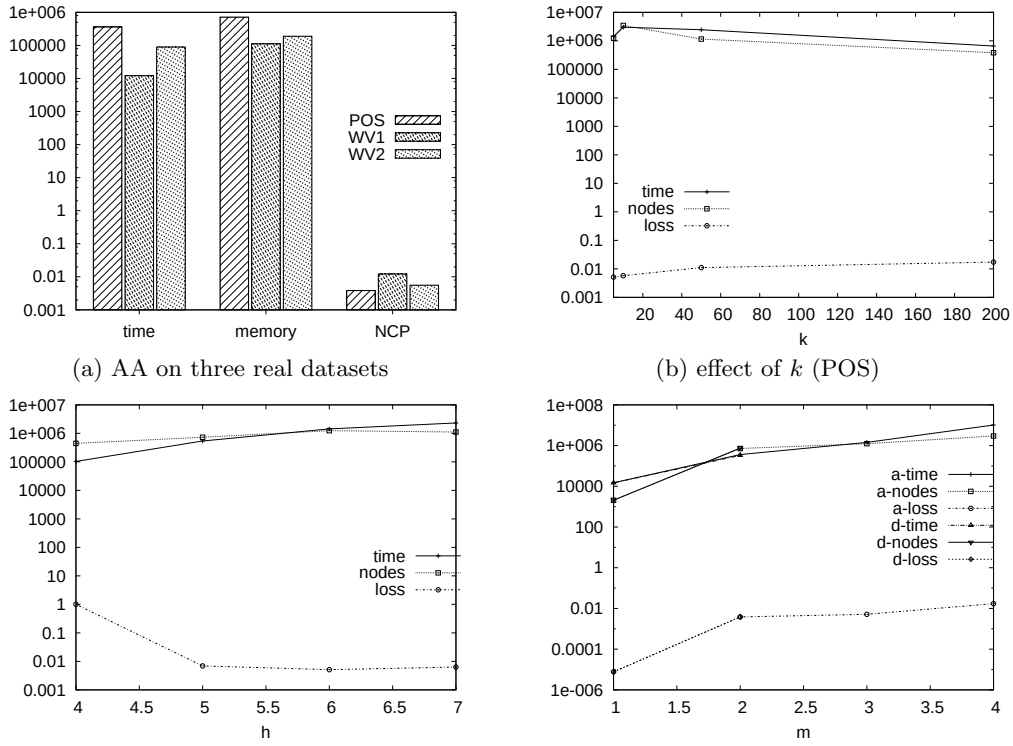
[7] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. of ACM SIGMOD*, pages 1–12, 2000.

[8] V. S. Iyengar. Transforming Data to Satisfy Privacy Constraints. In *Proc. of SIGKDD*, pages 279–288, 2002.

[9] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient Full-domain k-Anonymity. In *Proc. of ACM SIGMOD*, pages 49–60, 2005.

[10] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian Multidimensional k-Anonymity. In *Proc. of ICDE*, 2006.

[11] N. Li, T. Li, and S. Venkatasubramanian. t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In *Proc.*



(a) AA on three real datasets

(b) effect of k (POS)

(c) effect of h (POS)

(d) effect of m (POS)

Figure 11: Apriori algorithm on the original datasets

of ICDE, pages 106–115, 2007.

[12] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-Diversity: Privacy Beyond k-Anonymity. In *Proc. of ICDE*, 2006.

[13] A. Meyerson and R. Williams. On the Complexity of Optimal K-anonymity. In *Proc. of ACM PODS*, pages 223–228, 2004.

[14] M. Nergiz, C. Clifton, and A. Nergiz. Multirelational k-anonymity. Technical Report CSD TR 08-002.

[15] M. Nergiz, C. Clifton, and A. Nergiz. Multirelational k-anonymity. In *Proc. of ICDE*, pages 1417 – 1421, 2007.

[16] M. E. Nergiz and C. Clifton. Thoughts on k-anonymization. *Data and Knowledge Engineering*, 63(3):622–645, 2007.

[17] H. Park and K. Shim. Approximate algorithms for k-anonymity. In *Proc. of ACM SIGMOD*, pages 67–78, 2007.

[18] P. Samarati. Protecting Respondents' Identities in Microdata Release. *IEEE TKDE*, 13(6):1010–1027, 2001.

[19] L. Sweeney. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[20] V. S. Verykios, A. K. Elmagarmid, E. Bertino, Y. Saygin, and E. Dasseni. Association Rule Hiding. *IEEE TKDE*, 16(4):434–447, 2004.

[21] X. Xiao and Y. Tao. Anatomy: Simple and Effective Privacy Preservation. In *Proc. of VLDB*, pages 139–150, 2006.

[22] 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.

[23] Q. Zhang, N. Koudas, D. Srivastava, and T. Yu. Aggregate Query Answering on Anonymized Tables. In *Proc. of ICDE*, pages 116–125, 2007.

[24] Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *Proc. of KDD*, pages 401–406, 2001.