Anonymizing Collections of Tree-Structured Data

Olga Gkountouna, Student Member, IEEE, and Manolis Terrovitis

Abstract—Collections of real-world data usually have implicit or explicit structural relations. For example, databases link records through foreign keys, and XML documents express associations between different values through syntax. Privacy preservation, until now, has focused either on data with a very simple structure, e.g., relational tables, or on data with very complex structure e.g., social network graphs, but has ignored intermediate cases, which are the most frequent in practice. In this work, we focus on tree structured data. Such data stem from various applications, even when the structure is not directly reflected in the syntax, e.g., XML documents. A characteristic case is a database where information about a single person is scattered amongst different tables that are associated through foreign keys. The paper defines $k^{(m,n)}$-anonymity, which provides protection against identity disclosure and proposes a greedy anonymization heuristic that is able to sanitize large datasets. The algorithm and the quality of the anonymization are evaluated experimentally.

Index Terms—privacy, tree data, anonymity, structural knowledge, generalization, disassociation.

1 INTRODUCTION

As personal information is collected in increasingly detailed level by companies and organizations, privacy related concerns are posing significant challenges to the data management community. Data anonymization techniques have been proposed in order to allow processing of personal data without compromising user’s privacy. Nevertheless, practical problems like dependencies between values in personal records do not have a satisfying solution. In this paper, we focus on the anonymization of tree-structured personal records where values are linked through structural links.

Personal information rarely comprises just a single tuple in modern information systems. The information concerning a single individual usually spans over several tables or it is kept in a more flexible representation as an XML record. Such tree structured data cannot be anonymized effectively with table based anonymization methods since the structural relation between different fields substantially differentiates the problem. The problem of anonymizing tree structured data has only been addressed in existing research literature, in the context of multirelational $k$-anonymity [35]. In our approach we consider a more general case for tree structured data and we propose an anonymization method that does not rely solely on the generalization of values, but also on the simplification of the data tree.

Consider the medical records of Fig. 1. The depicted trees at the top represent two medical records. Each tree branch describes a health related incident. The first level after root holds information about the hospital where a client was admitted. The children nodes of the hospital nodes show the diagnosis. An attacker who knows that a person $X$ suffered from “Gastritis” and that $X$ was admitted to “Hospital$_1$” cannot distinguish between the two trees. If the attacker also knows that $X$ was treated for “Gastritis” at “Hospital$_1$”, then he can be certain that the top left record is the medical record of $X$. To prevent attackers who have such background knowledge from associating records to individuals we provide an anonymization method that offers protection against identity disclosure. We focus on identity disclosure for three main reasons: a) in many practical cases there are strict utility requirements that cannot be met when more powerful guaranties are applied, b) there is often inability to characterize attributes as sensitive or non-sensitive and c) the privacy protection law in most countries usually focuses on identity [4], [3], [5], [2], [1].

The paper proposes $k^{(m,n)}$-anonymity, which guarantees that an attacker who knows up to $m$ elements of a record and to $n$ structural relations between the $m$ elements will not be able to match her background knowledge to less than $k$ matching records in the anonymized data. The anonymization procedure does not only generalize values that participate in rare item combinations but also simplifies the structure of the records. The simplification is performed by removing nodes from long paths and creating new smaller paths. Returning to the example of Fig. 1, we can ensure that both records will be indistinguishable to an attacker who knows that a patient was treated for “Gastritis” in “Hospital$_1$”, by placing “Gastritis” as a sibling to the hospital (as shown in the two trees in lower part of Fig. 1). By examining these records, an attacker can infer that
both these patients were treated for “Gastritis” and that they have both been to “Hospital1” and “Hospital2”. The information that one patient was treated for gastritis in the particular hospital is hidden, so the attacker can no longer distinguish between the two records based on her background knowledge.

We propose two anonymization algorithms in this direction. Our first AllCutSearch (ACS) algorithm explores in a top-down fashion the lattice of all possible combinations of value generalizations, and for each different generalization it explores the possible structural transformations, and finds a solution that satisfies $k^{(m,n)}$-anonymity. Because of the large solution space ACS cannot scale to large collections of data and thus we propose a more aggressive greedy heuristic (GCS) which prunes the solution search-space by selecting on-the-fly the most promising candidate solutions. Our experimental evaluation shows that GCS scales well with the size of the dataset, and finds a solution very close to the one found by ACS in most tested cases. Our main contributions include the following:

- We define the problem of anonymizing tree structured data and we explain in detail how the record structure can act as a quasi-identifier.
- We define the $k^{(m,n)}$-anonymity privacy guarantee and explain how it is efficient in concrete attack scenarios.
- We introduce a novel data transformation, structural disassociation, which simplifies the structure of the records and provides more flexibility to the anonymization procedure.
- We propose a novel anonymization algorithm and a new information loss metric that takes into account both structural and value generalizations.
- We experimentally evaluate the proposed anonymization method, compare it to multirelational $k$-anonymity [35], and demonstrate that it manages to provide anonymized datasets with limited information loss.

2 PROBLEM DEFINITION

2.1 Data Model.

We consider a collection $D$ of records that have a tree structure with nodes which take values from a domain $I$. Each record $t$ corresponds to a different individual. The root of each tree is a pseudo-id indicating a different individual and all other nodes indicate a characteristic value of the individual. We do not consider duplicate sibling nodes or order between siblings, so our trees are unordered attribute trees. All records follow a common schema that defines the class of each node, e.g. the element in the case of XML. Each class $A$ has a domain $I_A$, domains of different classes are mutually exclusive $I_A \cap I_B = \emptyset$ for $i \neq j$. The union of all class domains is $I$. A path from the root to the leaves cannot have more than one nodes of the same class. The schema of the trees defines a partial order for the node classes. Following a path from the root, we progressively meet nodes of larger class, i.e., a class $B$ is larger than $A$ if we meet values of class $B$ after values of class $A$. The ordering of classes is arbitrary and it is defined by the publisher. The intuition between the ordering is to avoid logical discrepancies in the structure of the tree; if records include paths in the form of “Hospital$\rightarrow$Disease$\rightarrow$Treatment”, as in Fig. 1, then there will be no records where the information will appear in different order, e.g., “Disease$\rightarrow$Hospital$\rightarrow$Treatment”.

The proposed anonymization methods address datasets like $D$. The original data owned by the publisher might be in a different form, e.g., a multirelational schema, but it has to be transformed to a dataset with the structure of $D$ for the anonymization procedure. Richer information schemas e.g., graphs, references from one tree to another, are not addressed and can lead to privacy breaches.

2.2 Attack model.

We consider attackers who have partial knowledge about a person, i.e., they know a part of the information that exists in her record, and they want to use this partial knowledge to identify the complete record in $D$. We assume that an attacker has only positive knowledge about values and structural relations for any user record. We do not consider attackers who have negative knowledge i.e., the fact that a user is not associated with a certain value. We consider only one structural relation: the relation of ancestor-descendant (we denote that $a$ is a descendant of $b$, i.e., the attacker might know that two nodes appear in the same path. Finally, we assume that attackers can have structural knowledge only about nodes whose values are known to them: if an attacker knows $m$ values of a record $\{v_1, \ldots, v_m\}$, her structural knowledge would be a set of ancestor-descendant pairs of values from $\{v_1, \ldots, v_m\}$.

The attacker can use her background knowledge of node values and structural relations to filter the records. If the matching records are few, then there is a privacy breach.
Assume that the dataset of Fig. 2 is published and that an attacker knows that the target individual was treated for flu in Hospital_1 and that he was prescribed antibiotics in Hospital_2. Using this knowledge the attacker seeks a record that contains a path “Hospital_1 −→ Flu” and a path “Hospital_2 −→ Antibiotics”. As shown in Fig 3 this matches only record r3 from Fig. 2, so the attacker identifies the record of the target individual as r3.

2.3 Privacy Guarantee

We propose a new privacy guarantee that protects the identity of the individuals who are associated with tree records from attackers with the aforementioned capabilities by extending the $k^{(m,n)}$-anonymity guarantee [44] to address structural knowledge. $k^{(m,n)}$-anonymity guarantees that any attacker who knows up to $m$ elements of a record, will not be able to identify less than $k$ records in the published data. We define $k^{(m,n)}$-anonymity as:

Definition 1: ($k^{(m,n)}$-anonymity guarantee) A tree database $D$ is considered $k^{(m,n)}$-anonymous if any attacker who has background knowledge of $m$ node labels and $n$ structural relations between them (ancestor-descendant), is not able to use this knowledge to identify less than $k$ records in the published data.

An important characteristic of $k^{(m,n)}$-anonymity guarantee is the assumption that any node or structural association can be used as a quasi identifier. This is a very different assumption from the one made in $k$-anonymity where it is apriori known which parts of a record can act as a quasi identifier. When everything can act as quasi identifier, $k$-anonymity will create identical records as in the case of $k$-anonymity. As generalization and suppression, $k$-anonymity introduces a large information loss to the resulting dataset and at the same time it provides little gain in terms of privacy; records are anonymized to protect identification from attackers who have complete knowledge of them, i.e., from attackers who know all record values. $k^{(m,n)}$-anonymity is motivated by this observation, and assumes that while attackers might know any part of a record, it is unlikely that they will know the complete record. Moreover, if some attacker actually knows the complete record, there is no point in preventing them from identifying it in the dataset, since there is nothing additional to be revealed. The most important goal is to be able to prevent record identification by attackers who have partial knowledge. The parameterized nature of $k^{(m,n)}$-anonymity allows the publisher to tune the protection levels to their needs. As a result, $k^{(m,n)}$-anonymity allows anonymizing data with significantly reduced information loss with respect to $k$-anonymity and scales gracefully to highly dimensional data.

2.4 Anonymization Operations

A tree dataset $D$ can be transformed to a dataset $D'$ which complies to $k^{(m,n)}$-anonymity, by a series of transformations. The key idea is to replace rare values with a common generalized value and to remove ancestor-descendant relations when they might lead to privacy breaches.

2.4.1 Generalization

We assume the existence of a data generalization hierarchy (DGH) for every item of $I$. Each value of a class $A$ is mapped to a value in the next most general level and these values can be mapped to even more general ones. All class hierarchies have a common root denoted as “$*$”, which means “any” value and is equivalent to suppressing the value, as in Fig. 6. The proposed anonymization procedure adopts a global recoding approach towards generalizations. When a value is generalized, then all its appearances in the dataset are replaced by the new, generalized value. Moreover, when a value is generalized then all its siblings are generalized to the same item. We refer to such replacements made by the anonymization algorithm as generalization rules, e.g., {Gastritis, Diarrhea}−→{Stomach Disorder}, “Gastritis” and “Diarrhea” are replaced by “Stomach Disorder”.

Fig. 3. Example of an attack scenario.

Fig. 4. 3(2,1)-anonymous dataset of Fig. 2.

Fig. 5. 3-anonymous dataset of Fig. 2.
Disorder”. The anonymization algorithm will identify a generalization cut \( C \) on the DGH. A generalization cut defines the generalization level for each item in the data domain \( \mathcal{I} \), i.e., it defines a horizontal “cut” on the hierarchy tree. For example, the horizontal cut shown in Fig. 6 implies the generalization rules: \{Flu, Bronchitis\} \rightarrow \{Lung Disease\} and \{Pain killers, Antibiotics\} \rightarrow \{Medicine\}. The cut includes only the highest nodes under the dotted line in the figure.

We note that the adopted data model assumes that sibling nodes are always distinct. When the initial nodes are generalized, different sibling values can be replaced by a common generalized one, e.g., in Fig. 4 “Flu” and “Bronchitis” have been generalized to “Lung Disease”. To comply with the data model we merge the two appearances of “Lung Disease” and their subtrees, i.e., all paths under “Flu” and “Bronchitis” appear now under “Lung Disease”.

### 2.4.2 Structural Disassociation

Value generalization cannot address the structural background knowledge of the attacker to provide \( k^{(m,n)} \) anonymity. For the latter, we need to hide the ancestor-descendant relationships that are rare enough to be identifying. We call this operation structural disassociation SD:

**Definition 2:** (structural disassociation) Let \( p = a \rightarrow \cdots \rightarrow b \rightarrow n_1 \rightarrow \cdots \rightarrow l \) be a path in the previous path would result to two paths \( a \rightarrow \cdots \rightarrow b \rightarrow n_2 \rightarrow \cdots \rightarrow l \) and \( r \rightarrow \cdots \rightarrow b \rightarrow n_1 \rightarrow \cdots \rightarrow l \), which share the common prefix \( r \rightarrow \cdots \rightarrow b \).

As in the case of value generalization we opt for global recoding in the case of structural disassociation; if we disassociate \( a \rightsquigarrow b \) relation, this operation will take place in all records of \( D \). In the anonymized data there will be no occurrences of \( a \rightsquigarrow b \).

Consider the example of Fig. 7. The original record \( r_2 \) of Fig. 4 reveals that the patient was treated for gastritis in Hospital2, but in the anonymized data this relation is lost; the recipient of the data knows that the patient was treated for gastritis but does not know at which hospital.

Note that the children nodes of the node “Gastritis” remain as children of its parent node, i.e., “Antibiotics” becomes a direct child of “Hospital2”.

**2.5 Information Loss**

The value generalization and the structural disassociation transformations distort the original data and introduce information loss to the published anonymized data. To evaluate the effect of the anonymization procedure we need a common metric both for value generalizations and for structural disassociations. Our basic idea is to measure the reduced expressivity of the anonymized trees. To this end, we have opted for a simple metric the reverse path domain (RPD), which captures the reduction in the domain of generalized and structurally disassociated paths.

Assume that we have a path \( p = a_1 \rightarrow b_1 \rightarrow c_1 \), where \( a_1,b_1,c_1 \) are original terms with class domains \( A,B,C \). Then original path domain \( \mathcal{I}_p \) is \( \mathcal{I}_p = A \rightarrow B \rightarrow C \) with size \( |\mathcal{I}_p| = |A| \times |B| \times |C| \). Assume in the anonymization process we generalize \( a_1 \) to \( A \) and that \( C(A) \) is the number of distinct values that exist in the same level of generalization with \( A \), in the same class. Then, the path \( p \) is transformed to a generalized path \( p_g = A \rightarrow b_1 \rightarrow c_1 \). The size of the domain \( \mathcal{I}_{p_g} \) of \( p_g \) is \( |\mathcal{I}_{p_g}| = |C(A)| \times |B| \times |C| \). Then RPD is this case is:

\[
\text{RPD}(p_g) = \frac{1}{|\mathcal{I}_{p_g}|} = \frac{1}{|C(A)| \times |B| \times |C|} \quad |\mathcal{I}_p| = |A| \times |B| \times |C|
\]

where \( d() \) is a function that gives the depth of a node, i.e., its distance from the root. The intuition behind the \( d() \) factor is that nodes that are closer to the root are more important. This intuition was verified experimentally. The RPD for a random path \( p = u_1 \rightarrow \cdots \rightarrow u_n \) is:

\[
\text{RPD}(p) = \frac{1}{|\mathcal{I}_p|} = \frac{1}{|C(u_1)| \times \cdots \times |C(u_n)|} \quad |\mathcal{I}_p| = |A| \times |B| \times |C|.
\]

The RPD for a tree record \( t \) is defined as the average RPD of all its distinct paths from the root to a leaf \( p \):

\[
\text{RPD}(t) = \frac{1}{|\mathcal{L}_t|} \sum_{p \in \mathcal{L}_t} \text{RPD}(p)
\]

where \( |\mathcal{L}_t| \) is the number of nodes in \( t \). Note, that we need not take into account the structural disassociation transformations explicitly, since they implicitly affect RPD(\( t \)). For example, assume that in \( p = a_1 \rightarrow b_1 \rightarrow c_1 \) the relation \( b_1 \rightsquigarrow c_1 \) has to be removed to ensure privacy. Then we would end up with two paths \( p_1 = a_1 \rightarrow b_1 \) and \( p_2 = a_1 \rightarrow c_1 \) with the common prefix \( a_1 \). It is easy to see that the RPD of any of these two paths is greater than the RPD of the original path, i.e., \( \text{RPD}(p) < \text{RPD}(p_1) \) and \( \text{RPD}(p) < \text{RPD}(p_2) \), so \( \text{RPD}(p) \) will also be smaller than the average of \( \text{RPD}(p_1) \) and \( \text{RPD}(p_2) \). Structural disassociation always increases the value for RPD(\( t \)).

**Example 1:** Consider the generalization hierarchy of Fig. 6. The RPD of \( r_1 \) and \( r_2 \) of Fig. 4 are:

\[
\text{RPD}(r_1) = \frac{1}{3} \cdot \left( \frac{1}{1+2+6} + \frac{1}{1+2+6} + \frac{1}{1+2+6} \right) = 0.03
\]

\[
\text{RPD}(r_2) = \frac{1}{2} \cdot \left( \frac{1}{2+6+3} + \frac{1}{2+6+3} \right) = 0.023
\]

The RPD for a dataset \( D \) is defined as the average RPD of all its tree records \( t \):

\[
\text{RPD}(D) = \frac{1}{|D|} \sum_{t \in D} \text{RPD}(t)
\]

where \(|D|\) is the total number of records in the dataset.
2.5.1 Multi Level Mining Loss ($ML^2$)

Capturing the information loss that is caused by anonymization is a challenging task since the anonymized dataset might be used for different types of analysis. This is a problem common to every anonymization framework, and does not have a single best answer. To provide a comprehensive understanding of the $k^{(m,n)}$-anonymity’s impact on the quality of the data, we employ multiple information loss metrics that capture different aspects of the data information. The Multi Level Mining Loss ($ML^2$) [45], [21] is used in literature to express the amount of information that is lost when mining frequent itemsets of different generalization levels of the anonymized data. Given a dataset $D$ and its anonymous version $D'$, the $ML^2$ of $D'$ is one minus the ratio of the number of frequent itemsets that are preserved in $D'$, to the number of frequent itemsets in $D$, mined at all generalization levels.

In the tree data scenario, we are interested in preserving structural information as well as attribute values. $ML^2$ can be adapted to focus on the mining of frequent subtrees.

$$ML^2 = 1 - \frac{\sum_{i=0}^{d} FT_i(D')}{\sum_{i=0}^{d} FT_i(D)}$$

where $FT_i()$ is the number of frequent subtrees, at the $i$-th generalization level. We calculate $FT_i(D)$ and $FT_i(D')$ as follows. We project the original data to DGH levels (i.e., cuts which contain values of the same generalization level) and perform frequent subtree mining at each level $i$. The total number of all these itemsets is $\sum_{i=0}^{d} FT_i(D)$. We follow the similar procedure for $D'$, but in this case we cannot project any generalized item of $D'$, to a more refined generalization level, than its current. Intuitively, $ML^2$ measures the percentage of frequent subtrees that were lost in the released data, due to anonymization transformations.

**Example 2:** Assume that “r→Hospital$_2$→Flu” is a frequent subtree in the original data. Then “r→General Hospital→Lung Disease” and “r→Hospital→Disease” are also frequent subtrees at the next two generalization levels. These three frequent subtrees will contribute to the denominator of $ML^2$. Assume that the anonymization process results in a cut that contains the values “Hospital$_2$”, “Lung Disease”. In this case, the frequent subtree “r→Hospital$_2$→Flu” will not be found in the anonymized data. On the other hand, the frequent subtrees “r→General Hospital→Lung Disease” and “r→Hospital→Disease” will be mined since their values are above the generalization cut, so 2 will be added to the nominator of $ML^2$. If the path “Hospital$_2$→Lung Disease” had been structurally disassociated, then none of these original frequent subtrees could be mined in the result data.

2.5.2 Differential Multi Level Mining Loss ($dML^2$)

While $ML^2$ computes only the number of frequent subtrees that no longer appear in their exact initial form in the anonymized data, it is also important to measure the similarity distance between the frequent subtrees mined in the original data and the frequent subtrees mined from the anonymized data. To this end, we use an adjusted version of the Differential Multiple Level Mining Loss ($dML^2$) [45]:

$$dML^2 = \frac{\sum_{i=0}^{d} \sum_{ft \in FT_i(D)} dtree(ft | D \rightarrow D')}{\sum_{i=0}^{d} FT_i(D)}$$

where $FT_i(D)$ is the set of frequent subtrees $ft$ in the original data, projected at the $i$-th Generalization level, and function $dtree(ft | D \rightarrow D')$ calculates the distance between a subtree $ft$ existing in the original data and its anonymous version in the released dataset $D'$, taking equally into account value generalizations and structural dissimilarities. More formally $dtree$ is defined as follows:

$$dtree(ft | D \rightarrow D') = 0.5 \frac{\sum_{v \in ft} dlevel(v, v')}{N(ft)} + 0.5 BR(ft)$$

where $N(ft)$ is the number of nodes in frequent subtree $ft$, $BR(ft)$ is the number of ancestor-descendant relations between nodes of $ft$, and $dlevel(v, v')$ returns the difference between the generalization levels between the initial value of node $v$ and its anonymization $v'$, divided by the DGH height.

**Example 3:** Returning to the mined trees of Example 2, we would calculate the value of $dtree$ for the frequent subtree “r→Hospital$_2$→Lung Disease”, given the initial subtree “r→Hospital$_2$→Flu” as $0.5 \cdot \frac{1/3}{3} + 0.5 \cdot 0 = 0.083$, since DGH height is 3 and the difference between the levels of “Flu” and “Lung Disease” is 1. Had the relation “Hospital$_2$→Lung Disease” been structurally disassociated, the value of $dtree$ would be $0.5 \cdot \frac{1/3}{2} + 0.5 \cdot \frac{1}{3} = 0.583$.

3 Anonymization Algorithm

The problem we want to solve is the following: Given a dataset $D$, the parameters of $k^{(m,n)}$-anonymity and a generalization hierarchy $H$, we want to transform by generalization and structural disassociation the dataset $D$ to a dataset $D'$ for which $k^{(m,n)}$-anonymity holds and the information loss is minimum. The solution to this problem is a pair $(C, SD)$ of a generalization cut $C$ and a set of structural disassociation rules $SD$. By applying $(C, SD)$ to $D$ we get a $k^{(m,n)}$-anonymous dataset $D'$. Due to the high difficulty of the problem (we show that it is NP-Hard), we propose a heuristic algorithm that finds a “good” (local optimum) but not optimal solution. To capture the information loss, we use an estimation function based on $RPD$, the $RPD_a$, which we define later in the section.

For each generalization cut we have multiple different structural dissimilarities. The complete solution space comprises all the combinations of generalization cuts and structural disassociation transformations. Finding the optimal solution is NP-Hard. This follows from the fact that the problem of finding the optimal $k$-anonymization of a relational table, which is known to be NP-Hard [32], can be reduced to a specific case of the $k^{(m,n)}$-anonymization of tree records. Assume that a relational table $R$ is represented as a collection $D$ of tree structured records that all have the same structure: a root node and all the fields of the table as direct children of the root. Finding the optimal $k^{(m,n)}$-anonymization for $D$, for $m, n$ equal to $R$’s arity solves...
also the problem of finding the optimal $k$-anonymity for $R$. Since the latter is an NP-Hard problem, then so is the identification of the optimal $k^{(m,n)}$-anonymization.

Because of the problem complexity we avoid performing an exhaustive search of the solution space, instead we provide a heuristic that explores promising subspaces. We propose a top-down algorithm which initially considers all values to be generalized to “$*$”. The algorithm traverses the generalization hierarchy from top to bottom by specializing one node at each iterative step. Each specialization creates a new hierarchy cut. If the cut does not guarantee $k^{(m,n)}$-generalization hierarchy from top to bottom by specializing values to be generalized to “$*$”. The algorithm traverses the proposed subspaces. We do not need to project every record and then create the synopsis tree for cut $C$. We can directly project $S$ to $C$ and create the projected synopsis tree $S_C$.

The projection procedure is done as follows:

- A new entry is added to $L$ for every generalized item $g_i$ that appears. This new entry has a list of $id$ which is the result of the union of the $id$ lists of all items $i$ that are mapped to $g_i$. We create the list of sidelinks associated with $g_i$, as the set of all sidelinks in the entry of every item $i$ that is mapped to $g_i$.
- The label $i$ of every node of $S$ is replaced by the generalized item $g_i$ defined in $C$.
- Sibling nodes with the same $g_i$ label are merged together. The new merged node has the same label as the original nodes, and its list of $id$ is the union of the lists of all original nodes. Redundant sidelinks from item $g_i$ of the list to the same merged node of the tree are eliminated.

Example 5: Consider the synopsis tree of Fig. 8, and the cut $C$=\{Lung disease, Gastritis, Diarrhea, Neurological, Orthopedic, Medicine, Hospital1, Hospital2, Rea, Gaia\}. $C$ implies generalization rules \{Flu, Bronchitis$\Rightarrow$Lung disease and \{Antibiotics, Painkiller$\Rightarrow$Medicine. The respective projected tree $S_C$ and sidelsenk list $L$ are illustrated in Fig. 9. Since nodes “Flu” and “Bronchitis” were siblings under “Hospital1” in the synopsis, their projected nodes are merged as one “Lung disease” node in the projected tree. This node has the $id$ list $[2,3,4]$ which is the union of lists of “Flu” and “Bronchitis” under Hospital1 in the synopsis.

**The RPD as a heuristic.** The information loss metrics defined in Section 2.5 are used to evaluate the quality of the final results and they are calculated over the raw data. RPD is the average RPD of every record of the dataset, $ML^2$ and $dML^2$ require mining the original and the anonymized dataset. Using them for evaluating every candidate solution would lead to an impractical anonymization algorithm. Instead, our algorithm uses a computationally cheap heuristic, which is based on $RPD$ but it is calculated based only on the Projected Synopsis tree $S_C$. To compensate for this, after some experimental testing, we take into account the support of each node and also the number of distinct nodes ($nids_{S_C}$) that exist in $S_C$. The approximate $RPD_a$ for a path $p$ from the root of $S_C$ to a leaf is given by the following function:

$$RPD_a(p) = \frac{\text{sup}(u_1) + \cdots + \text{sup}(u_n)}{\text{d}(u_1) \times |C(u_1)| \times \cdots \times \text{d}(u_n) \times |C(u_n)|}$$

where $\text{sup}(u_i)$ is the support of the node $u_i$. The $RPD_a$ of the synopsis tree includes all information of the input dataset in a compressed form. It is sufficient for calculating efficiently the support of combinations of original items and paths. In the process of anonymization we need to create a synopsis tree for every projection of $D$ to a cut $C$. Fortunately, we do not need to project every record and then create the synopsis tree for cut $C$. We can directly project $S$ to $C$ and create the projected synopsis tree $S_C$. The projection procedure is done as follows:

### 3.1 Synopsis Tree

The synopsis tree facilitates deciding on the $k^{(m,n)}$-anonymity of a dataset by tracing not only the support of item combinations from $I$, but also the support of paths that contain them. The term support refers to the number of records that contain the path. The Synopsis tree is a form of trie tree, similar to FP-tree [23] and has two main parts:

- **A tree structure.** which is created by superimposing all records of $D$. Every record’s root node is mapped to a single node, the root $r_s$ of the synopsis tree. All paths that appear in a record are superimposed to the synopsis tree starting from $r_s$. Each node $n$ has two elements: a) a label representing the item that is mapped to it and b) a sorted list of the ids of all the records that contain the exact path from the root to the current node, i.e., $r_s \rightarrow \cdots \rightarrow n$.

- **An array $L$, with one entry for each item $i$ of $I$.** Each entry has three elements a) a label with the item $i$ that corresponds to the entry, b) a list of the ids of all records that contain $i$ and c) a link to every node in the tree that is labeled with $i$. These links are marked with dashed lines in Fig. 9, and will be referred to as sidelinks in the rest of the paper. The order of the items in the array is not important; it depends on the insertion order. The array allows a horizontal access to the synopsis tree and it also supports checking whether the $k^{m}$-anonymity holds, which is a prerequisite of $k^{(m,n)}$-anonymity, without traversing the tree. Note, that keeping the list of record ids that are associated with item $i$ in each entry of $L$ is redundant; the list can be created by merging the lists that are associated with the nodes that have an $i$ label. Because checking the support of combinations of items is a very common operation in the anonymization procedure we opted to keep the redundant list, in order to increase performance.

**Example 4:** Consider the tree dataset of Figure 2 which contains four records. The respective synopsis tree is illustrated in Figure 8. “Hospital1” appears in records 1, 2 and 3, as a child of the root node in all of them. Thus, in the synopsis-tree a node labelled “Hospital1” appears as a child of the root. Its list of $id$s is [1, 2, 3] as shown in the figure.
Fig. 9. Projected synopsis tree of Figure 8

for the whole $S_C$ is given by Equation 7:

$$RDP_a(S_C) = \frac{1}{ndss_C} \times \frac{1}{lvs_C} \sum_{p \in S_C} RDP_a(p)$$

where $ndss_C$ is the total number of nodes of $S_C$.

### 3.2 Candidate Solution Check

The projected synopsis tree $S_C$ and the sideline list $L$ of a dataset $D$, can be used to quickly verify if a solution $(C, SD)$ (data hierarchy cut $C$ and structural disassociation rules $SD$) are sufficient for providing $k^{(m,n)}$-anonymity when applied to $D$. This process is performed in two phases: the generalization check and the structural relation check. The former will examine whether all itemsets of size $m$ contained in $D_C$ appear at least $k$ times. The latter will examine whether there are in $D$ at least $k$ records, that contain the $m$-sized combinations, when also considering any $n$ structural relations between them. The pseudo code for the ValueCheck and StructureCheck functions appear in Algorithms 1 and 2 respectively.

Property 1: ValueCheck returns true if and only if $D_C$ is $k^{(m,n)}$-anonymous, i.e., every combination of $m$ values appears at least $k$ times.

If the generalization check is successful, the projected synopsis tree $S_C$ is created and it is used by StructureCheck to examine whether $D_C$ supports more than $k$ times a combination of items $cnr$ with $cnr$ relations between them. StructureCheck takes as input $S_C$, i.e., the projected tree to $C$, the sideline list $L$, a combination of items $cmn$, and a set of relations $cnr$ that hold between the $cmn$ items. In Lines 4-6 the algorithm collects the list ($clist$) of records that support relation $cmn \rightarrow cnr$ in $D_C$ and then intersects this list with the list ($clist$) of records that support the rest of the relations in $cnr$ in Line 8. If at any point the $clist$ contains between 0 and $k$ records then the algorithm terminates, as the solution violates the $k^{(m,n)}$-anonymity.

Property 2: StructureCheck returns true if and only if the items $cmn$ with $cnr$ relations between them, appear at least $k$ times in $D_C$.

ValueCheck and StructureCheck are not completely symmetric; ValueCheck checks whether a cut $C$ provides $k^{(m,n)}$-anonymity to $D_C$, i.e., it checks all $m$-sized combinations of the items appearing in $C$. On the other hands, StructureCheck checks only one combination of items $cmn$ and one set of relations between them $cnr$. The reasons behind this choice are explained in the anonymization algorithm of the following section.

### 3.3 Anonymization algorithm

We propose a top-down algorithm that explores the solution space starting from a state where all nodes are generalized to the root of the hierarchy tree (a single node labelled “*”), and no structural disassociations have been taken place ($SD = \emptyset$), and then proceeds by considering less generalized cuts and structural disassociation rules for the projected dataset. The complete solution space for the problem comprises of all possible cuts and all possible disassociation rules for them. Exhaustively examining all possible solutions is not practical even for small datasets. Instead we propose a heuristic algorithm, named ACS (Algorithm 1 Cut Search), that examines generalizations and structural disassociation symmetrically; it exhaustively examines every generalization cut, but then greedily chooses how to structurally disassociate each projected dataset. We chose this strategy because of the respectively asymmetric cost in examining all generalization cuts and all disassociation rules for the projected dataset.

To understand how ACS works consider the cut generalization graph of Figure 10. The graph nodes depict all possible cuts and the edges show how one cut can be specialized to another one, by only specializing one item in $C$. The ACS will start from the most generic cut (*) and

**Input:** $S_c, L, (cmn, cnr)$ ($cmn$ items, and $cnr$ relations between them)

**Output:** true, false ($true$ if $D_C$ contains $(cmn,cnr)$ $k$ times, else false)

1. $clist = \emptyset$ (first intersection will initialize it)
2. for all $\{an \rightarrow dn\}$ relations of $cnr$ do
3. $tlist = \emptyset$
4. for all nodes $dn$ in the tree do
5. if the path from $dn$ to the root contains an then
6. $tlist = tlist \cup tlist$ // all trees that adhere to $an \rightarrow dn$
7. remove $\{an \rightarrow dn\}$ from $cnr$
8. $clist = clist \land tlist$;
9. if $clist$ has 0 items then
10. return true
11. else if $clist$ has less than $k$ items then
12. return false
13. return true

**Algorithm 1.** ValueCheck()

**Algorithm 2.** StructureCheck()
visit all cuts of the graph only once. The pseudocode for ACS is presented in algorithm 4.

The algorithm uses a stack $STK$ to keep all neighbor nodes that have not been examined yet. At each step the first node of the stack is popped and the algorithm examines whether the cut $cCut$ can provide $k^{(m,n)}$-anonymity. For the algorithm examines whether simple $k^{m}$-anonymity holds, by invoking ValueCheck. If ValueCheck fails, then $k^{(m,n)}$-anonymity cannot be achieved with this cut, or any cut that is more specialized than the current one, so the algorithm simply continues with the next item of $STK$. If ValueCheck succeeds, then it is certain that the current cut can lead to $k^{(m,n)}$-anonymity by performing enough structural disassociations. Thus the algorithm creates the projected synopsis tree $SC$ and invokes $FixStructure$ (depicted in algorithm 3), which performs the required structural disassociations $cSD$. If the cost of the solution ($cCut, cSD$) is smaller than the cost of the best solution found until now, then ($cCut, cSD$) is stored as the best solution. The algorithm then inserts all children of $cCut$ to $STK$ and marks them as closed. When there are no more nodes in $STK$ the algorithm terminates and outputs $(C, SD)$ as the best solution. The cost (in terms of information loss) of each solution is estimated using the RPD metric, introduced in Section 2.5.

Example 6: Consider the DGH of Figure 6. ACS would first generalize all values to $\{\star\}$, as shown in cut $C_0$ of Figure 10. Then ACS proceeds to the next cut $C_1 \{\text{Disease, Treatment, Hospital}\}$. In the next step there are three possible values to generalize, which correspond to the three child nodes of $c_1$. After checking for $k^{(m,n)}$-anonymity, these cuts are ordered from lower to higher loss cost. Let the order be $c_2$, $c_3$, $c_4$ as shown in Figure 10. Cut $c_2$ is specialized first. This results to four new candidate cuts to be checked and ordered by their cost. If at least one of them satisfies $k^{(m,n)}$-anonymity and has a lower cost than $c_2$, ACS proceeds to specialize it, and so forth. Otherwise, we roll back to $c_3$, which has three possible new specializations as “Disease” is now closed. If $c_2$, $c_3$ and $c_4$ didn’t satisfy $k^{(m,n)}$-anonymity, ACS would roll back to $c_0$ and terminate.

The pseudocode for the $FixStructure$ function is presented in Algorithm 3. For each combination of items $cnn$ of the current cut, $FixStructure$ considers all possible combinations of relations $cnr$ of sizes up to $n$ and calculates the supports of $cnn$ under $cnr$ using the projected synopsis tree $SC$ and the sidelinks of $L$. If any combination of $cnr$ relations leads to a breach of $k^{(m,n)}$, anonymity, $FixStructure$ structurally disassociates the relations of $cnr$, starting from the least frequent and adds the disband transactions to $SD$, until the $cnn$ items, under the $cnr \setminus SD$ relations, are supported at least $k$ times. Since all item combinations of size $m$ are checked and all sets of relations between them up to size $n$ are checked, the final solution $(C, SD)$ will guarantee $k^{(m,n)}$-anonymity for $D$.

Note that in the actual implementation we do not keep track of the closed cuts explicitly. Instead we keep track of the values that have been specialized, as they are fewer and easier to represent. Once a value has been specialized (i.e.,...
the structural disassociations \(SD=\{\text{Hospital}_1\leftarrow\text{Gastritis}, \text{Hospital}_2\leftarrow\text{Gastritis}\}\) and guarantees that any attacker who knows 2 values and the relation between them cannot identify less than 3 records.

### 3.4 Greedy Cut Search Algorithm

The AllCutSearch (ACS) algorithm avoids exploring the whole solution space, but can still be quite expensive if the data domain or the dataset is large. To deal with bigger and more expressive datasets, we propose the Greedy Cut Search Algorithm GCS, which performs a partial best first traversal of the generalization cut graph. The GCS works as the ACS algorithm, but instead of examining all child cuts of the current cut \(ccut\) it examines only the \(g\) lowest cost ones (the \(STK\) is a priority queue with lowest RPD cuts being first). At each iteration step, GCS pops all siblings of the first node from \(STK\) (Line 9 of algorithm 4) but does not initially insert any child of the popped cuts to the \(STL\). It first examines each cut, and then inserts the children only of \(g\) cuts that have the lowest cost. This way the algorithm greedily follows the most promising paths and can significantly reduce the search space and computational time. Experimental results show that even for a small \(g\), its results are almost identical to those of ACS.

**Example 7:** Returning to the example of Figure 10 and assuming \(g=2\), GCS would first generalize all values to \(c_0\) and then proceed to \(c_1\). In the next step, GCS would check the three new candidate cuts for \(k^{(m.n)}\)-anonymity, and order them from lower to higher RPD, e.g., \(c_2, c_3, c_4\). GCS would add \(c_2\) and \(c_3\) to the priority queue, but not \(c_4\) since it has greater cost and \(g=2\). The new cuts would be added to the GCS priority queue and the lowest cost one, e.g., \(c_2\) would be popped. Specializing \(c_2\) results to 4 new cuts, but GCS would only add to the priority queue on the 2 cuts that have the least RPD out of the 4 candidates. When the priority queue becomes empty, GCS terminates.

### 3.5 Discussion

**Complexity.** We do not provide a complete complexity analysis due to space restrictions and also because of the difficulty of calculating the number of all possible cuts. Still, there are several complexity results that are drawn based on the ValueCheck and StructureCheck. The size of the dataset \(|D|\) affects the construction of \(S\), the size of the lists that are associated to the nodes and the operations on the lists. In all these cases the effect of \(|D|\) is linear, thus the algorithm is \(O(|D|)\). FixStructure and ValueCheck have to calculate every combination of \(m\) items from the current cut \(C\) (in the worst case \(C=I\)), resulting to a complexity of \(O(\sum_{i=1}^{m} m!)\). Moreover, \(I\) together with DHG affect the total number of possible cuts.

In practice the effect of \(m\) and \(I\) differs significantly from the worst case. In the case of \(m\) the observed running time is not exponential to \(m\); it actually reduces up to some value of \(m\) and then increases almost linearly. This is due to the fact that as \(m\) affects the anonymization procedure in two competing ways: as \(m\) grows each solution check is more expensive, but at the same time it is harder to find a satisfying solution to the problem. The top down algorithm exploits the latter factor and it examines fewer and simpler solutions as \(m\) grows. The size of \(I\) does not have a significant impact in practice; even if \(I\) grows, the algorithm is not affected if it does not reach solutions that are in the bottom of the solution graph, e.g., the graph of Figure 10. Finally \(k\), does not directly affect any algorithm but it affects how many solutions will satisfy the guarantee, so large values of \(k\) limit the number of solutions that will be examined by the algorithms.

**l-diversity** If sensitive values, which cannot act as quasi identifiers, are identified apriori as in most \(l\)-diversity approaches [48], [20], then we can easily extend our algorithm to provide \(l\)-diversity. The basic change needed is to add an additional condition in ValueCheck and StructureCheck that would require the \(k^{(m.n)}\)-anonymous groups of trees to also be \(l\)-diverse, i.e., to also contain \(l\) well-represented sensitive values [31].

**Negative knowledge** When the data are sparse, like tree data, negative knowledge is less important and dangerous than positive knowledge; there are a few values associated with an entity but numerous that are not. However, there could be cases where negative knowledge can be acquired by an attacker and used to attack an anonymized dataset. If negative knowledge has to be completely covered, i.e., every item that does not appear in the record must be considered, then adjusting our heuristic for providing simple \(k\)-anonymity would be the best choice. Since negative knowledge is not as identifying and easy to acquire as positive knowledge, the most interesting and practical case, arises when we take it partially into account. For example, we could take into account only negations of hospitals (because an attacker might be able to infer that a patient has not visited a hospital that is very far from her place of residence) but ignore negations of treatments, since it is a lot harder for an attacker to verify that a patient never received a specific kind of treatment. The proposed framework can easily address intermediate cases; we would only have to populate the records with the selected negations, e.g., “not Hospital”. The algorithms would then provide protection against attackers who know \(m\) values that appear or do not appear in a record, considering only a subset of the possible negative knowledge. This way, we can address the most probable and dangerous negative knowledge without introducing significant additional information loss.

### 4 Experimental Evaluation

In the section we present the experimental evaluation of our algorithms. All implementations were done in C++ and all experiments were performed on an Intel Core i7 CPU, with 6GB RAM, running Ubuntu Linux.

**Algorithms.** We implemented and compared 4 algorithms, including ACS and GCS that are described in Section 3. We implement a third anonymization algorithm that does not perform any structural transformations on the data. Instead, it rejects any solution that would require structural
disassociations in order to achieve $k^{(m,n)}$-anonymity. We term it as OnlyCutSearch (OCS) and use it as a point of reference to understand better the impact of structural disassociation in the anonymization procedure. We further implemented the most closely related method to our own, MiRaCle [34], a local recoding generalization algorithm which clusters tree-like multiRelational records to form $k$-anonymous groups. Miracle uses only generalization (local recoding) and suppression to transform the original dataset.

Data. For the experimental evaluation of the proposed algorithms we use the data from TPC-H [6], which is a typical example of a database of customers, orders, products and suppliers, all linked via foreign keys. We parse the relational tables and use the foreign keys to create tree records that represent different individual customers. The resulting trees express the following information: each customer has made a number of orders at a particular date, each containing a number of products (items). To simplify the experimental evaluation we used only the attributes: customer nation, order price, order date, item quantity, manufacturer and brand name from the relational tables, and kept the structural relations between values implied by the schema of the database. Using the TPC-H data generator we created the datasets described in Table 1. We first created a dataset of 1M records and sampled it to create the two other ones. We limited the fanout of the records (each customer may have up to two orders, each containing up to three items) to create a dataset where the ratio of the size of each record to the total dataset size is small (if the records are too big and too detailed compared to the size of the collection, only very low quality anonymization can be produced with any method). We created a synthetic DGH of the values of the attributes with an average fanout of 5.

Parameters. We study the behavior of the algorithms with respect to the following parameters: a) $k$ parameter which controls the strength of our privacy guarantee, b) $m$ which quantifies the attacker’s knowledge, c) $n$ which measures the structural information that can be used as a quasi-identifier, d) the dataset size $|D|$ and e) parameter $g$ of the GCS algorithm. In every experiment we vary one of these parameters keeping others fixed. The default setting of our parameters is $k = 20$, $m = 3$, $n = 2$, $|D| = 250,000$ and $g = 2$. After some experimentation we have identified the best values for the climit and threshold parameters of MiRaCle as 150 and 0.1 and respectively.

Evaluation Metrics. We evaluate our method with respect to execution time and information loss in terms of the RPD, $ML^2$ and $dML^2$ metrics.

ACS vs. GCS. The first series of experiments, aims at investigating the performance differences between ACS and the aggressive heuristic of GCS. Fig. 11 shows the behavior of GCS as $g$ increases from 1 to 4. For $g = 2$, the performance of GCS in terms of information loss (measured by RPD here) converges with that of ACS. This finding is verified by the experiments of Fig. 12. The performance in terms of information loss is similar; in most cases both algorithms find the same solution. On the other hand, there is a huge gap in the performance of the two algorithms in terms of computational cost. As shown in Fig. 13 GCS is at least an order of magnitude faster in most settings. For the experiments of Fig. 12 and 13 we use the dataset of size 100K, since the computational cost of ACS greatly increases for larger datasets. Results show that GCS provides almost the same quality of anonymization with ACS at only a fraction of the computational cost. Because of this, we focus in the rest of the experimental evaluation on GCS.

Performance of GCS In Figures 14, 15 and 16 we evaluate the performance of GCS in terms of anonymization quality. We compare our results to both MiRaCle and OCS which rejects any possible solution that requires structural changes. The experiments show that GCS preserves better the data utility compared to algorithms that do not apply structural transformations on the data.

In Fig. 14 we see the performance of the three algorithms in terms of RPD. Despite the fact that MiRaCle uses local recoding, it cannot surpass the proposed algorithms. It introduces greater RPD in every case and for $m < 5$ the RPD for MiRaCle is double than the RPD for GCS. The inferior performance of MiRaCle in terms of utility is attributed to two factors: a) the relaxed guarantee offered by GCS and ACS and b) the structural disassociation they employ. As expected, ACS has a higher information loss than GCS for all values of the parameters. The performance gap between GCS and ACS is more significant for more relaxed guarantees; as $k$ decreases from 300 to 5 their difference increases from 8.7% to 39.5%. A decrease in $m$ from 6 to 4 also increases OCS’s information loss up to 54% higher than GCS’s.

The main advantage of GCS is evident when we consider the attacker’s structural knowledge. For $n = 0$ both GCS and ACS produce the same anonymization and outperform MiRaCle by 56%. As $n$ increases GCS remains relatively stable, while ACS increases by 53%. For $n = \{2, 3\}$ OCS’s loss is 1.5 times higher than GCS’s. MiRaCle remains stable, but it has already greatly reduced the data quality.

As $|D|$ increases from 100K to 1M records, GCS manages to reduce the information loss, by exploiting its increased flexibility in data transformations, while the information loss of ACS slightly increases. MiRaCle’s loss is relatively steady and is on average double the loss of GCS. Figures 15 and 16 show the experimental results of $ML^2$ and $dML^2$ metrics respectively. To measure them

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>No of records $</td>
</tr>
<tr>
<td>Avg nodes per record</td>
</tr>
<tr>
<td>Tree records depth $d$</td>
</tr>
</tbody>
</table>

| Attributes | 6 |

| Data Domain | 276 |

| GCS Fanout | 5 |
we mined the original and the anonymized datasets for frequent itemsets in all generalization levels, using support threshold 1%. This means that we mine frequent subtrees which appear in at least 2,500 records.

The results in terms of $ML^2$ and $dML^2$, demonstrate similar behavior as those of RPD. In the case of $ML^2$, Miracle manages to outperform in some cases both OCS and GCS. Still, this happens only when the anonymized result is of poor quality (for $m \geq 5$, half the average record size) and $ML^2$ is over 0.8, which means that 80% of the frequent subtrees are lost. On the other hand for $m < 5$, GCS manages to produce good quality results, but introducing 40% to 50% less information loss than Miracle. For example, in the case of the 250k dataset 10845 frequent subtrees were mined in the original data, at all generalization levels. While OCS preserves only 1864 of them, MiRaCle preserves 2358 subtrees. Both are outperformed by GCS which preserves 6457 subtrees for our default parameter setting ($k=20$, $m=3$, $n=2$).

$dML^2$ results show that even when the exact patterns cannot be mined in the anonymized dataset (the $ML^2$ is around 80% for Miracle and high for the other algorithms) the difference of those mined from the original ones is quite low; the respective $dML^2$ is below 19%. Here both OCS and GCS are clear winners over Miracle, which applies high level generalizations and suppressions. Again GCS proves almost insensitive to $n$, while the information loss for OCS significantly increases as $n$ grows.

**Execution Time**. The computational cost of our GCS algorithm is shown in Fig. 17. The effect of $k$ is depicted in Fig. 17(a). Higher values of $k$ allow GCS to prune a significant part of the search space and significantly reduce the computational cost. The execution time falls by 97% as $k$ goes from 5 to 100 and is further reduced by another 85.8% when $k$ increases to 300. The $m$ parameter affects the execution time in two competing ways as described in Section 3.5. This results to a local minimum for $m = 4$. Fig. 17(b) indicates that when $m$ rises from 3 to 4, execution time decreases by -70.5%, whereas after 4 it increases significantly. Parameter $n$ does not limit the search space of our algorithm, but it defines the amount of structural relations an attacker may know. Thus, execution time increases with $n$ as shown in Fig. 17(c) since the number of nodes combinations that are checked for privacy violations only increases. In Fig. 17(d) we see that GCS evaluation time increases linearly with the dataset size $|D|$.

**Negative Knowledge**. To support our hypothesis that negative knowledge does not constitute a significant danger in most real cases, we performed the following experiment: for each combination of $m$ values and $n$ relations, we randomly chose 1-4 additional values, which we assume as negative knowledge of the attacker. We assumed attackers who know the negation of values from the original domain (we mark results concerning these attackers as “original”) and attackers who know the negation of generalized values, from the next level of generalization (we mark those as “generalized”). In the case of attackers who know only negations of the original data, the attacks are easily thwarted; when the negated value has been generalized, it can no longer be used by the attacker, e.g., an attacker who knows “not flu”, cannot rule out a record containing “lung disease”, which is the generalization of “flu”. We chose the values in a way that the background knowledge of the attacker, both positive and negative, matches at least 1 record in the dataset. In the first two graphs of Fig. 18, we depict the number of individuals whose privacy has been breached, i.e., the number of individuals that can be identified by knowledge combinations which have support less than $k$ in the anonymized data, when the additional negative knowledge is considered. We report the average number of individuals whose privacy has been breached per positive knowledge combination. In the first graph we depict how this number changes for different $k$ and in the second one how it scales when the size of the negative knowledge $q$ increases. We note that even when considering negative knowledge of 4 values from the original domain, less than 0.001 individuals per knowledge combination is affected, and when considering generalized values this number rises to 0.21. In the next two graphs of Fig. 18, we investigate the vulnerability degree of the exposed individuals. We depict the support distribution of
the problematic knowledge combinations, i.e., the combinations whose support becomes less than $k$. The $y$-axis traces the percentage of the total combinations that fall in each size bucket. In the $x$ axis we depict the negative knowledge that was possessed by the attacker in each case. In the third graph of Fig. 18, the attacker only knows the negation of original items, and she cannot reduce the support of a combination to less than $[10-14]$ records (with $k=20$), even for $q=4$, and this only happens in the 0.001% of the combinations. When negation of generalized values is considered, the attacker can reach combinations of support $[3-4]$ but, this happens only for $q=4$ and only in the 0.0058% of the combinations. In summary, we believe that the results support our hypothesis that negative knowledge does not give substantial de-anonymization power to an attacker in sparse multidimensional data.

5 RELATED WORK

L. Sweeney proposed $k$-anonymity guarantee to address linking attacks [40]. A table is $k$-anonymous if each record is indistinguishable from at least $k - 1$ others with respect to the QI set [39], [40]. To achieve this, QIs are transformed to form groups of records with identical QI values, called equivalence classes. The two most popular techniques to achieve this, generalization and suppression were introduced in [40].

Generalization based techniques that consider only global recoding to limit the search space were explored in [9], [26]. Limiting the search space comes at the cost of increased information loss. Generalization techniques that are based on multidimensional local recoding [27], [7], [49] manage to achieve lower information loss. We employ a global single-dimensional subtree-domain recoding approach, as we explain in Section 2. The loss of utility due to global recoding is compensated by the limits that parameters $m$ and $n$ set on attackers knowledge, which result to lower information loss.

The objective of most anonymizing algorithms is to find an optimal recoding of the data that satisfies a given privacy guarantee and preserves as much data utility as possible. The latter is accomplished by minimizing a function which estimates the information loss. [32] 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 with the approximation bound $O(k \cdot \log k)$. [8] improved this bound to $O(k)$, while [38] further reduced it to $O(\log k)$. 
Even though $k$-anonymity guarantees the protection against identity disclosure, sensitive information may be revealed when there are many identical sensitive attribute values within an equivalence class (attribute disclosure). The concept of $l$-diversity [31] was introduced to address the limitations of $k$-anonymity. [19], [48], [53], [30] present various methods to solve the $l$-diversity problem efficiently. [20] extends [48] for sparse high-dimensional data. [46] proved that optimal $l$-diversity is NP-hard for any $l \geq 3$, under the suppression model. They provide an $O(l \cdot d)$-approximation, where $d$ is the number of QI attributes in the released data. [24] showed the vulnerability of anatomy [48] in deFinetti attacks, which aim to learn the correlation between sensitive and non-sensitive attributes using a Bayesian network. However, [15] showed that the deFinetti attack is effective only for small values of $l$. [29] proposes an extension of $l$-diversity, called $t$-closeness, to protect against skewness attacks where the distribution of SA in an equivalence class is different than the distribution of SA in the whole dataset. In [37], the authors propose $k$-anonymity and $l$-diversity algorithms that minimize the number of data accesses by utilizing the summary structure maintained by the database management system for query selectivity. [10] extend $t$-closeness by proposing $\beta$-likeness that limits the information gain of a sensitive value, which is defined as the difference between its original support and the respective support of this value in the microdata.

Perturbation-based methods add noise to the data to achieve privacy [41], [11]. They attempt to bound attacker’s posterior confidence about a sensitive value in relation to the prior belief. Other noise-adding methods enforce differential privacy [17], [47], [28], which guarantees that the presence or absence of any individual’s record in the released dataset does not substantially affect the result of query analysis. However, [25] showed that differential privacy does not adequately limit inference about the participation of an individual in the data. Interestingly, [15] has also shown that, even though any single individual is dominated by the noise, the noise is in turn dominated by the signal emerging from the whole population. Thus, a Naïve Bayes classifier can be built to infer individuals’ sensitive values with non-trivial accuracy. Recently, [16] studied empirical privacy and utility, based on the posterior beliefs of an attacker and their ability to draw inferences about sensitive values in the data, to compare different privacy models. They show that the difference between differential privacy and various syntactic models is less dramatic than previously thought. Especially when accuracy is considered important, syntactic methods are preferred, making a compromise between privacy and utility. Similarly, [14] compares and analyzes both approaches, and suggests that differential privacy is more appropriate for privacy-preserving data mining, while syntactic methods are suitable for privacy-preserving data publishing.

There are several works on privacy protection in high dimensional data. There is significant work on privacy preservation on graph databases [13], [54], [51], but the focus there is to protect the identity or other properties of a single node in a single large graph. There is also work on privacy protection in trajectories [43], [33], [52]. In [36] the authors use a clustering method based on a log cost metric to anonymize trajectory data. These works cannot easily be exploited in our setting, since it assumes attack scenarios and transformations tailored to spatiotemporal data. Furthermore, there is work in more similar settings to our that offers protection against attribute disclosure [20] or differential privacy [12].

The works of [44], [45], [22], [42], [50], [18] are closely related to our approach, as they provide similar guarantees for unstructured data. In [44] the notion of $k^m$-anonymity is introduced, which is similar to our $k^{(m,n)}$-anonymity, but covers attackers who have only values as background knowledge, so it is not suitable for our data model. [45] proposed the application of disassociation in set-valued data, where a transaction could be split in two or more parts. [50] and [18] also anonymize transaction data. All these methods do not assume any structural attacker knowledge, as they deal with unstructured data. Thus, they are not directly comparable to our method. [34], [35] consider the problem of providing privacy protection to individuals, whose data are scattered in several tables in a relational data base. To this end they propose multirelational k-anonymity. Still, the problem they address is simpler than the one we are facing: there is a distinction between sensitive values and QI and no structural transformation is considered. 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. The proposed algorithm in [34], [35], Miracle, employs local recoding for anonymizing the data. With respect to our proposal it provides a stricter privacy guarantee at the cost of increased information loss. Depending on the application area and the requirements, we believe that our proposal allows the data publisher to better balance the tradeoff between privacy and utility, as it is customizable and can provide more relaxed privacy guarantees.
6 CONCLUSIONS
In this paper, we are addressing the problem of anonymizing tree-structured data in the presence of structural knowledge. We propose \( k^{(m,n)} \)-anonymity privacy guarantee which addresses background knowledge of both value and structure. We present an anonymization algorithm which is able to create \( k^{(m,n)} \)-anonymous datasets, by employing value generalization and a novel data transformation, which we term structural disassociation. We demonstrate experimentally that the proposed greedy algorithm is able to scale to large datasets and outperform, in terms of information loss, methods that are based solely on value generalization.

REFERENCES

Olga Gkountouna is a PhD candidate in the National Technical University of Athens (NTUA, Greece), and a member of the Knowledge and Database Systems Laboratory’s group. Her main research interests lie in the areas of data privacy, data mining, and management of semi-structured data. She is a recipient of the ‘HERACLEITUS II’ EU-funded research fellowship.

Manolis Terrovitis is an associate researcher at the Institute for the Management of Information Systems (IMIS) of the Research and Innovation Centre in Informatics, Communication and Knowledge Technologies “Athena”. He received his PhD in 2007 from the National Technical University of Athens. His main research interests lie in the areas of data privacy, indexing and query evaluation.