ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

IMPLEMENTATION OF ANONYMIZING COLLECTIONS OF TREE-STRUCTURED DATA

Sandeep .N .L*, Dr. K. Thippeswamy
* Dept. of Computer science & Engg, VTU PG Studies, Mysore, Karnataka, INDIA.
Professor, Dept. of Computer science & Engg, VTU PG Studies, Mysore, Karnataka, INDIA.

KEYWORDS: privacy, tree data, anonymity, structural knowledge, generalization, disassociation.

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.

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. 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.

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

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

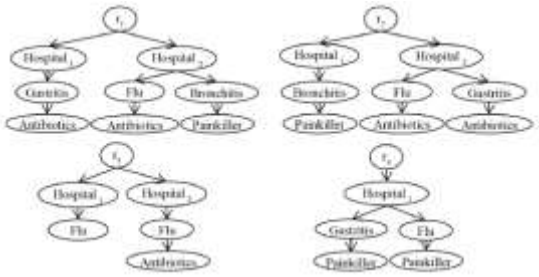

Fig. 1. Records of a medical tree database TD.

both these patients were treated for "Gastritis" and that they have both been to "Hospital$_1$" and "Hospital$_2$". 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, and demonstrate that it manages to provide anonymized datasets with limited information loss.

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 $I_{Ai} \cap I_{Aj} = \emptyset$ 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→Disease→Treatment", as in Fig. 1, then there will be no records where the information will appear in different order, e.g., "Disease→Hospital→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.

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

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 b is a descendant of a as a → 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.

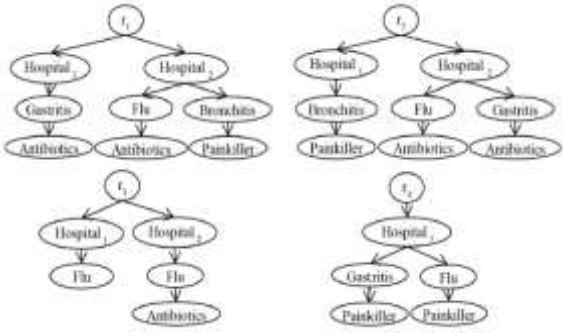

Fig. 2. Example dataset D.

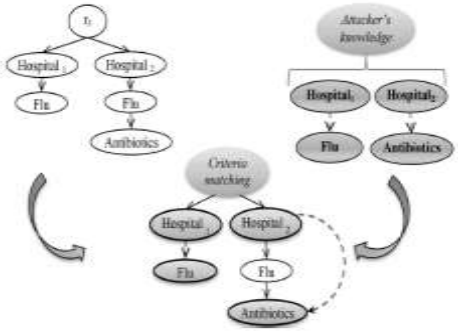

Fig. 3. Example of an attack scenario.

Assume that the dataset of Fig. 2 is published and that an attacker knows that the target individual was treated for flu in Hospital1 and that he was prescribed antibiotics in Hospital2. Using this knowledge the attacker seeks a record that contains a path "Hospital1 Flu" and a path "Hospital2 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-anonymity guarantee [44] to address structural knowledge. k^m-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 D.

An important characteristic of $k^{(m,n)}$-anonymity guarantee is the assumption that any node or structural association

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

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 Fig. 5. It depicts a 3-anonymization of the example of Fig. 2 that has been created using generalization and suppression. K-anonymity introduces a large information

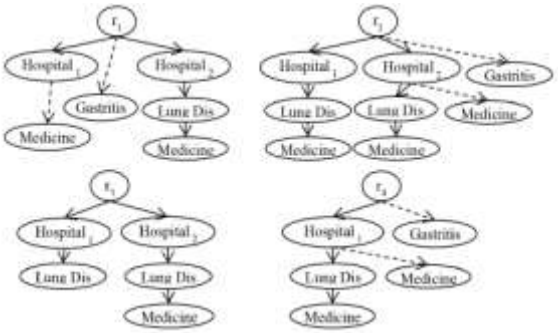

Fig. 4. $3^{(2,1)}$ -anonymous dataset of Fig. 2.

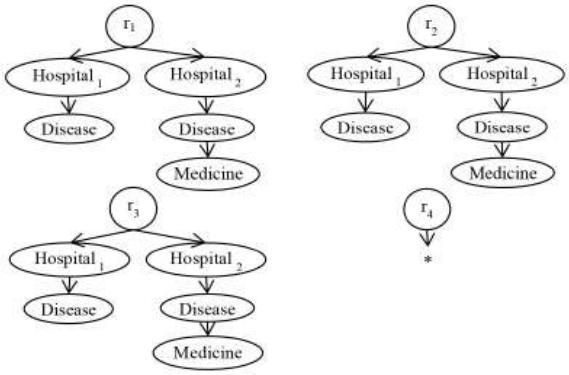

Fig. 5. 3-anonymous dataset of Fig. 2.

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. A $3^{(2;1)}$-anonymous version of the example of Fig. 2 appears in Fig. 4. Here, every attacker who knows up to 2 values related to a person and the relation between these two values, e.g., "X was treated for lung disease in $Hospital_1$" cannot identify less than 3 records in the published data.

2.4 Anonymization Operations

A tree dataset D can be transformed to a dataset D^0 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".

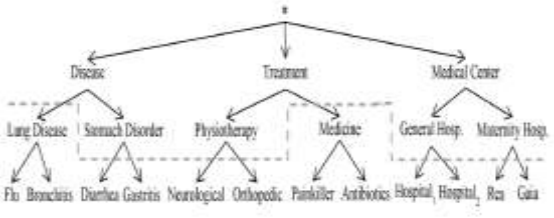

Fig. 6. A cut on the generalization hierarchy.

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 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}→{Lung Disease} and {Pain killers, Antibiotics}→{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 back-ground 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 be a path $r \rightarrow . . \rightarrow p_a \rightarrow a \rightarrow . . \rightarrow p_b \rightarrow b \rightarrow n_b \rightarrow . . \rightarrow l$. A structural disassociation of the relation a b in the previous path would result to two path $r \rightarrow . . \rightarrow p_a \rightarrow a \rightarrow . . \rightarrow p_b \rightarrow b \rightarrow n_b \rightarrow . . \rightarrow l$ and $r \rightarrow . . \rightarrow p_a \rightarrow b$, which share the common prefix $r \rightarrow . . \rightarrow p_a$.

As in the case of value generalization we opt for global recoding in the case of structural disassociation; if we disassociate a b relation, this operation will take place in all records of D. In the anonymized data there will be no occurrences of a b. Consider the example of Fig. 7. The original record r_2 of Fig. 4 reveals that the patient was treated for gastritis in $Hospital_2$, 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 "$Hospital_2$".

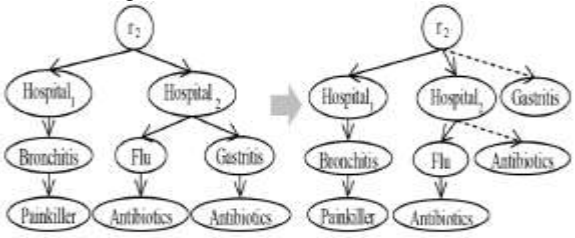

Fig. 7. Structural disassociation example.

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 (RP D), which captures the reduction in the domain of generalized and structurally disassociated paths.

ANONYMIZATION ALGORITHM

The problem we want to slove is the following: Given a dataset D, the parameter of $k^{(m,n)}$-anonymity and ma generalization hierarchy H, we want to transform by generalization and structural disassociation the dataset D to a dataset D^0 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^0. 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 RP D, the RP D_a, which we define later in the section.
For each generalization cut we have multiple different structural disassociations. 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 $k^{(m,n)}$-anonymity for the published dataset, then we explore the possible structural disassociations for this cut.

Checking whether a combination of a generalization cut and structural disassociation rules can provide $k^{(m,n)}$-anonymity if applied to D is not a trivial task. To perform the check efficiently we employ a memory structure termed synopsis tree, which we present in the following section.

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 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., $r_s \rightarrow . . \rightarrow n$.

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 Figure 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 tree 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. "$Hospital_2$" 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 labeled "$Hospital_2$" appears as a child of the root. Its list of ids is [1, 2, 3] as shown in the figure.

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

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:

A new entry is added to L for every generalized item gi 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 gi. We create the list of sidelinks associated with gi, as the set of all sidelinks in the entry of every item i that is mapped to gi.

The label i of every node of S is replaced by the generalized item gi defined in C.

Sibling nodes with the same gi 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 gi 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, $Hospital_1$, $Hospital_2$, Rea, Gaia}. C implies generalization rules {Flu, Bronchitis}→Lung disease and {Antibiotics, Painkiller}→Medicine. The respective projected tree S_C and sidelink list L are illustrated in Fig. 9. Since nodes "Flu" and "Bronchitis" were siblings under "$Hospital_1$" 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 $Hospital_1$ 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, M L^2 and dM L^2 require mining the original and the anonymized dataset. Using them for evaluating every candidate solution would lead to an impractical anonymization algorithm. In-stead, our algorithm uses a computationally cheap heuristic, which is based on RP D 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 (nds_{SC}) that exist in S_C .

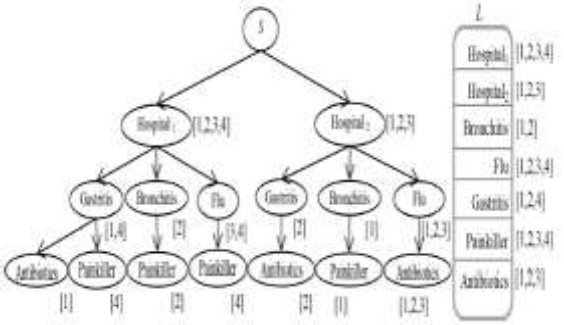

Fig. 8. Synopsis tree.

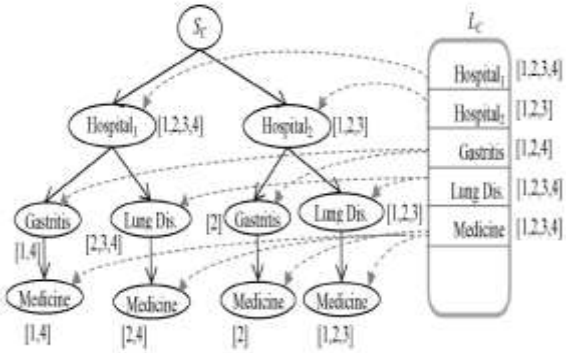

Fig. 9. Projected synopsis tree of Figure 8

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

3.2 Candidate Solution Check

The projected synopsis tree S_C and the sidelink 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.

ValueCheck examines whether there is a combination of items (with size less than or equal to m) that does not appear at least k times in D_C . Then C, SD cannot be a valid solution since C cannot create the required support for every combination. If the solution is discarded by the generalization check, we can avoid creating the projected synopsis tree S_C , which is expensive. It is easy to see that ValueCheck guarantees the following property:

Property 1: ValueCheck returns true if and only if D_C is k^m-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 cmn with cnr relations between them. StructureCheck takes as input S_C , i.e, the projected tree to C, the sidelink 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 (tlist) of records that support relation an dn in D_C and then intersects this list with the list

Input: C,L,m

Output: true, false {true if D_C is k^m-anonymous, else false}

```
1:  for all  cm combinations of size m in C do
2:     Intersect the lists from L for every item of cm
3:     if the intersection size is between k and 0 then
4:          return false
5: return true
```

Algorithm 1. ValueCheck()

(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 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-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 labeled "*"), and no structural disassociations have taken place (SD = Ø), 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 AllCutSearch (ACS) that examines generalizations and structural disassociation asymmetrically; it exhaustively examines very 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; the latter is significantly more expensive in realistic

datasets. 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 visit all cuts of the graph only once. The pseudocode for ACS is presented in algorithm 4.

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 = all ids {first intersection will initialize it}
2: for all {an dn} relations of cnr do
3:     tlist = Ø
4:     for all nodes dn in the tree do
5:         if the path from dn to the root contains an then
6:             tlist = dnUlist [ tlist //all trees that adhere to an → dn
7:     remove {an   dn} from cnr
8:     clist = clist ∩ 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 2. StructureCheck()

The algorithm uses a stack ST K 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 exam-ines whether the cut cCut can provide $k^{(m,n)}$-anonymity. First 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 with any cut that is more specialized than the current one, so the algorithm simply continues with the next item of ST K. If ValueCheck succeeds, then it is certain that the current cut can lead to $k^{(m,n)}$-anonymity by performing enough structural disassocations. Thus the algorithm creates the projected synopsis tree S_C 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 ST K and marks them as closed. When there are no more nodes in ST K 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 RP D metric, introduced in Section 2.5.

Example 6: Consider the DGH of Figure 6. ACS would first generalize all values to {*}, as shown in cut c_0 of Figure 10. Then ACS proceeds to the next cut c_1 {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 cmn of the current cut, FixStructure considers all possible combinations of relations cnr of sizes up to n and calculates the supports of cmn under cnr using the projected synopsis tree S_C 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 disbanded relations to SD, until the cmn items, under the cnrnSD 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.,

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

Input: S_C , L, C

Output: SD {disassociation rules}

```
1: SD = Ø
2: for all cmn combinations of m items from C do
3:    for i = 1 . . . n do
4:       for all cnr_i \ SD combinations of size i of the items of cmn do
5:       while not StructureCheck(S_C , L, cmn, cnr_i)  do
6:             select the relation a → b from cnr_i with the
               least positive support
7:             for all paths that contain a → b do
8:                move the children of b to become its siblings
9:                move b to become a sibling of a
10:            SD = SD U r //add r to existing disassociation rules
11: return SD
```

Algorithm 3. FixStructure()

Input: D, DGHierarchy, k, n, m

Output: (C, SD) {(C, SD) renders D $k^{(m,n)}$-anonymous}

```
1: Create synopsis tree S from D
2: Create inverted list L //L is created for generalized terms too
3: stack ST K = Ø
4: bestCost = ∞; //Minimum Loss
5: root = *
6: mark root as closed
7: ST K.push(root)
8: while ST K not empty do
9:    cCut=ST K.pop() //current cut cCut
10:   for all children C of cCut do
11:      if C not closed then
12:         mark C as closed
13:         ST K.push(C)
14:   if V alueCheck(cCut, S, L) then
15:      Create S_cCut //we project S to cCut
16:      cSD = F ixStructure(S_cCut, L, cCut)
17:      cCost = cost(cCut, cSD) //estimated cost of current solution
18:      if cCost < bestCost then
19:         bestCost = cCost
20:         (C, SD) = (cCU, cSD)
21: return  (C, SD);
```

Algorithm 4. AllCutSearch (ACS) Algorithm we examine a cut that contains its children in the DGH), the value is marked as closed in the following cuts. A cut is closed, i.e., redundant, when all its values are closed.

The trees in Figure 4 are the $3^{(2;1)}$-anonymous version of the initial dataset in Figure 2. The solution consists of the cut C={Lung Disease, Gastritis, Diarrhea, Neurological, Orthopedic, Medicine, $Hospital_1$, $Hospital_2$, Rea, Gaia} and

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

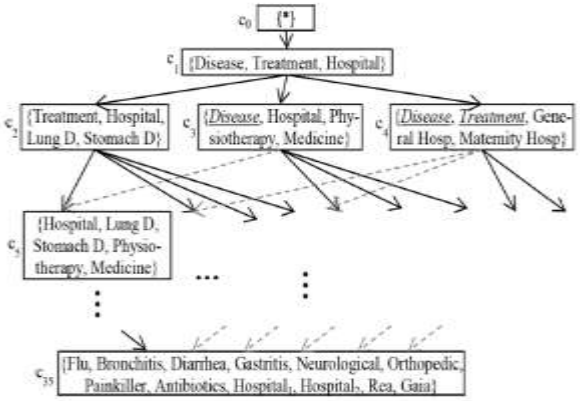

Fig. 10. Cut-enumeration Tree for the DGH of Figure 6

the structural disassociations SD={$Hospital_1$ Gastritis, $Hospital_2$ 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 ST K is a priority queue with lowest RP D_a cuts being first). At each iteration step, GCS pops all siblings of the first node from ST K (Line 9 of algorithm 4) but does not initially insert any child of the popped cuts to the ST L. 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 RP D_a, 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 RP D_a 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 $O(\frac{I^m}{})$. Moreover, I together m! 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.

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

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 $Hospital_2$". 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.

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

TABLE 1

No of records |D|	100k,250k,1M	Attributes	6
Avg nodes per record	12	Data Domain	276
Tree records depth d	4	DGH Fanout	5

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.

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

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. 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 RP D. Despite the fact that MiRaCle uses local recoding, it cannot surpass the proposed algorithms. It introduces greater RP D in every case and for m < 5 the RP D for MiRaCle is double than the RP D for GCS. The inferior performance of MiRaCle in terms of utility is attributed to two factors: a) the relaxed guarantee offered by GCS and OCS and b) the structural disassociation they employ. As expected, OCS has a higher information loss than GCS for all values of the parameters. The performance gap between GCS and OCS 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' 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 OCS produce the same anonymization and outperform MiRaCle by 56%. As n increases GCS remains relatively stable, while OCS increases by 53%. For n = f2, 3g OCS' 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 in-creased flexibility in data transformations, while the information loss of OCS slightly increases. MiRaCle's loss is relatively steady and is on average double the loss of GCS.

Evaluation Metrics. We evaluate our method with respect to execution time and information loss in terms of the RP D, M L^2 and dM L^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 RP D 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.

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

In Fig. 14 we see the performance of the three algorithms in terms of RP D. Despite the fact that MiRaCle uses local recoding, it cannot surpass the proposed algorithms. It introduces greater RP D in every case and for m < 5 the RP D for MiRaCle is double than the RP D for GCS. The inferior performance of MiRaCle in terms of utility is attributed to two factors: a) the relaxed guarantee offered by GCS and OCS and b) the structural disassociation they employ. As expected, OCS has a higher information loss than GCS for all values of the parameters. The performance gap between GCS and OCS 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' 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 OCS produce the same anonymization and outperform MiRaCle by 56%. As n increases GCS remains relatively stable, while OCS increases by 53%. For n = f2, 3g OCS' 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 OCS 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 ML2 and dML2 metrics respectively. To measure them we mined the original and the anonymized

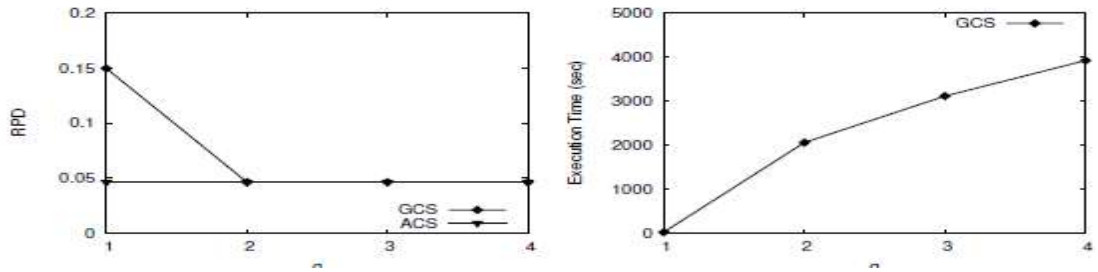

Fig. 11. Effect of parameter g.

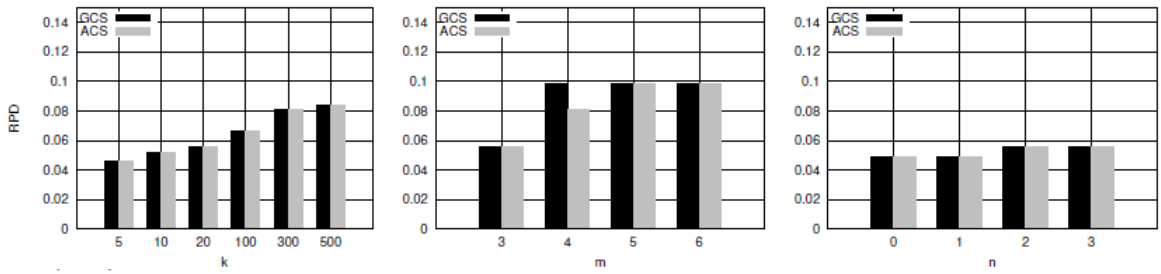

Fig. 12. Information loss: ACS vs. GCS.

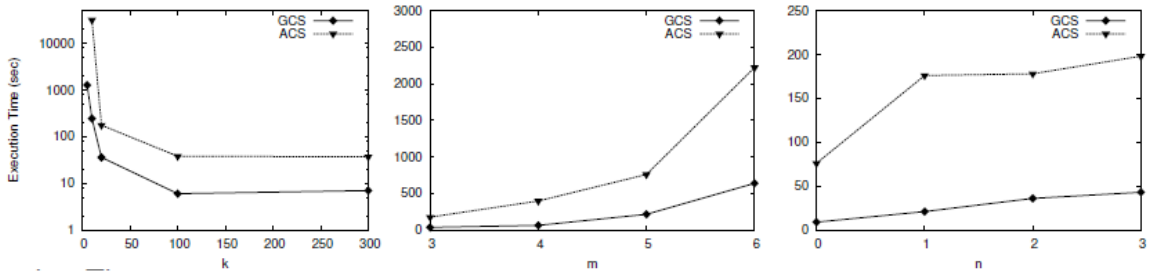

Fig. 13. Execution Time: ACS vs. GCS.

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 M L^2 and dM L^2, demonstrate similar behavior as those of RP D. In the case of M L^2, Miracle manages to outperform in some cases both OCS

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

and GCS. Still, this happens only when the anonymized result is of poor quality (for m 5, half the average record size) and M L^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).

dM L^2 results show that even when the exact patterns cannot be mined in the anonymized dataset (the M L^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 dM L^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

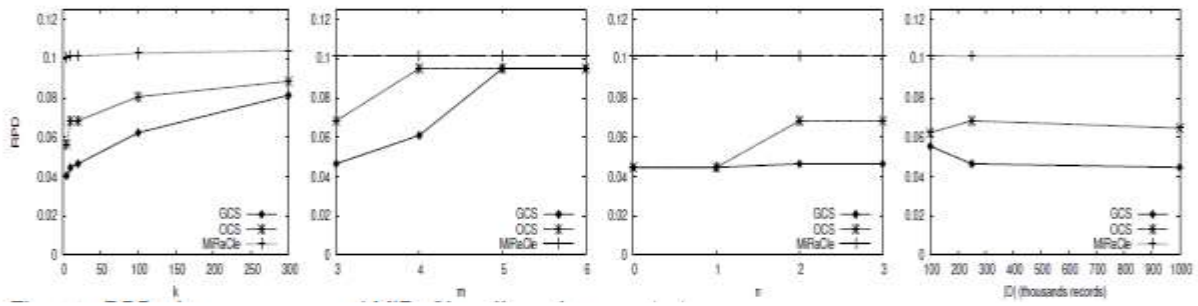

Fig. 14. RPD of GCS vs. OCS and MiRaCle: effect of k, m, n, |D|

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

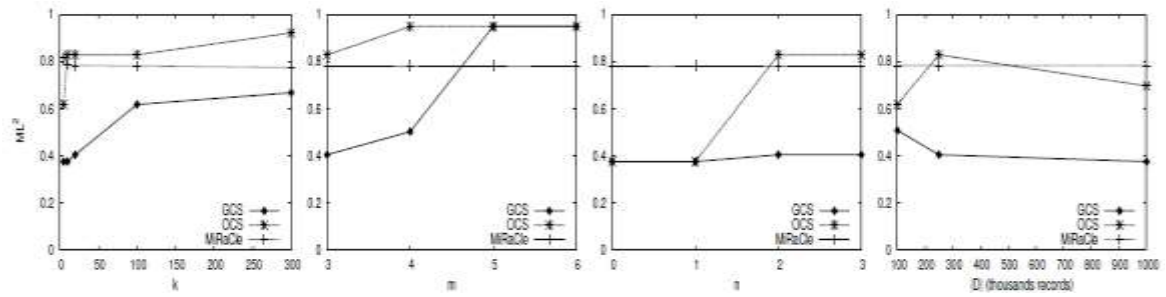

Fig. 15. ML2 Data mining loss metric.

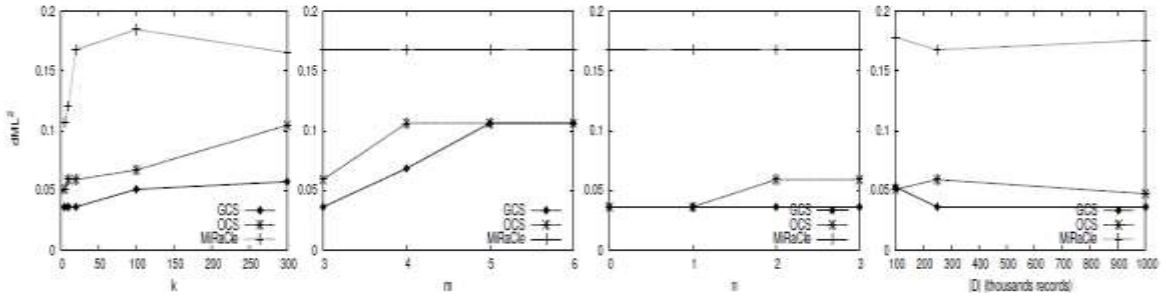

Fig. 16. dML2 Data mining loss metric.

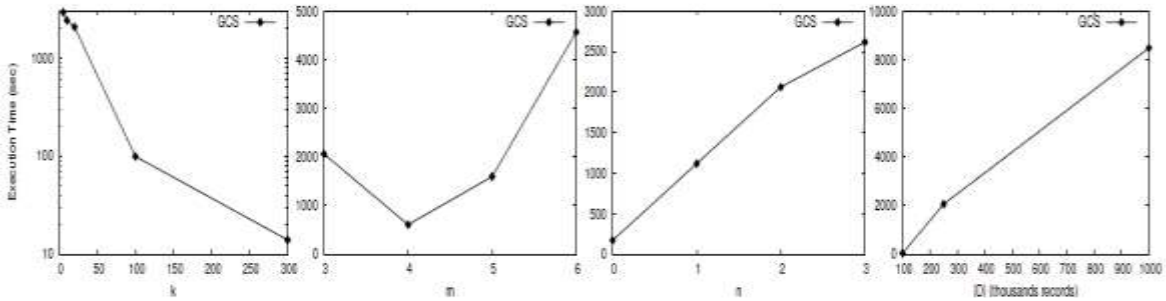

Fig. 17. Execution Time of GCS: effect of k, m, n, |D|

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.

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 logk). [8] improved this bound to O(k), while [38] further reduced it to O(logk).

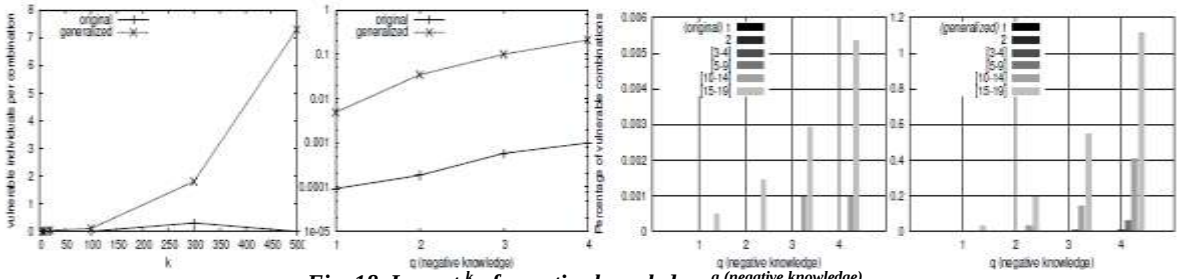

Fig. 18. Impact ^{k} of negative knowledge. $^{q\ (negative\ knowledge)}$

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 3, under the suppression model. They provide an O(l 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 -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 Naive 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. Sim-ilarly, [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] pro-posed 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.

CONCLUSIONS

In this paper, we are addressing the problem of anonymiz-ing tree structured data in the presence of structural knowl-edge. 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 experi-mentally 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

1. Australian Privacy Act. www.austlii.edu.au/au/legis/cth/consol act/ pa1988108.
2. Canadian Privacy Act. laws-lois.justice.gc.ca/eng/acts/P-21/.
3. Data Protection Act 1998, UK. www.legislation.gov.uk/ukpga/1998/ 29/contents.
4. GR Law. www.dpa.gr/portal/page? pageid=33,43560& dad=portal.
5. HIPAA act, US. http://health.state.tn.us/hipaa/.
6. TPC-H Homepage. http://www.tpc.org/tpch/.
7. G. Aggarwal, T. Feder, K. Kenthapadi, S. Khuller, R. Panigrahy, D. Thomas, and A. Zhu. Achieving Anonymity via Clustering. In PODS, 2006.
8. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Pani-grahy, D. Thomas, and A. Zhu. Approximation Algorithms for k-Anonymity. Journal of Privacy Technology, 2005.
9. R. J. Bayardo and R. Agrawal. Data Privacy through Optimal k- [45] M. Terrovitis, N. Mamoulis, and P. Kalnis. Local and global recoding Anonymization. In ICDE, pages 217–228, 2005.
10. J. Cao and P. Karras. Publishing microdata with a robust privacy guarantee. PVLDB, 5(11):1388–1399, 2012.
11. R. Chaytor and K. Wang. Small-domain randomization: Same privacy more utility. In VLDB, 2010.
12. R. Chen, N. Mohammed, B. C. M. Fung, B. C. Desai, and L. Xiong. Publishing set-valued data via differential privacy. PVLDB, 4(11):1087–1098, 2011.
13. J. Cheng, A. W.-c. Fu, and J. Liu. K-isomorphism: privacy preserving network publication against structural attacks. In SIGMOD, 2010.
14. C. Clifton and T. Tassa. On syntactic anonymity and differential privacy. In PRIVDB, 2013.
15. G. Cormode. Personal privacy vs population privacy: learning to attack anonymization. In SIGKDD, pages 1253–1261, 2011.
16. G. Cormode, C. Procopiuc, E. Shen, D. Srivastava, and T. Yu. Empirical privacy and empirical utility of anonymized data. In PRIVDB, pages 77–82, 2013.
17. C. Dwork. Differential privacy. In ICALP (2), pages 1–12, 2006.
18. G. Ghinita, P. Kalnis, and Y. Tao. Anonymous publication of sensitive transactional data. TKDE, 23(2):161–174, 2011.
19. G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis. Fast Data Anonymization with Low Information Loss. In VLDB, 2007.
20. G. Ghinita, Y. Tao, and P. Kalnis. On the Anonymization of Sparse High-Dimensional Data. In ICDE, 2008.

21. A. Gkoulalas-Divanis and G. Loukides. Utility-guided clustering-based transaction data anonymization. TDP, 5(1):223–251, 2012.
22. O. Gkountouna, S. Angeli, A. Zigomitros, M. Terrovitis, and Y. Vas-siliou. k^m-anonymity for continuous data using dynamic hierarchies. In PSD, pages 156–169. Springer, 2014.
23. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In SIGMOD, pages 1–12, 2000.
24. D. Kifer. Attacks on privacy and deFinetti's theorem. In SIGMOD, pages 127–138, 2009.
25. D. Kifer and A. Machanavajjhala. No free lunch in data privacy. In SIGMOD, pages 193–204, 2011.
26. K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient Full-domain k-Anonymity. In SIGMOD, 2005.
27. K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian Multi-dimensional k-Anonymity. In ICDE, 2006.
28. C. Li and G. Miklau. An adaptive mechanism for accurate query answering under differential privacy. PVLDB, 5(6), 2012.
29. N. Li, T. Li, and S. Venkatasubramanian. Closeness: A new privacy measure for data publishing. TKDE, 2010.
30. J. Liu and K. Wang. On optimal anonymization for l^+-diversity. In ICDE, pages 213–224, 2010.
31. A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-Diversity: Privacy Beyond k-Anonymity. In ICDE, 2006.
32. A. Meyerson and R. Williams. On the Complexity of Optimal Kanonymity. In PODS, pages 223–228, 2004.
33. A. Monreale, R. Trasarti, D. Pedreschi, C. Renso, and V. Bogorny. Csafety: a framework for the anonymization of semantic trajectories. TDP, 4(2):73–101, 2011.
34. M. Nergiz, C. Clifton, and A. Nergiz. Multirelational k-anonymity. In ICDE, pages 1417–1421, 2007.
35. M. Nergiz, C. Clifton, and A. Nergiz. Multirelational k-anonymity. IEEE TKDE, pages 1104–1117, 2009.
36. M. E. Nergiz, M. Atzori, Y. Saygin, and B. G¨uc¸. Towards trajectory anonymization: a generalization-based approach. TDP, 2(1):47–75, 2009.
37. M. E. Nergiz, A. Tamersoy, and Y. Saygin. Instant anonymization. ACM Trans. Database Syst., 36(1):2, 2011.
38. H. Park and K. Shim. Approximate algorithms for k-anonymity. In SIGMOD, pages 67–78, 2007.
39. P. Samarati and L. Sweeney. Generalizing Data to Provide Anonymity when Disclosing Information (abstract). In PODS (see also Technical Report SRI-CSL-98-04), 1998.
40. L. Sweeney. k-Anonymity: A Model for Protecting Privacy. IJUFKS, 10(5), 2002.
41. Y. Tao, X. Xiao, J. Li, and D. Zhang. On anti-corruption privacy preserving publication. In ICDE, 2008.
42. M. Terrovitis, J. Liagouris, N. Mamoulis, and S. Skiadopoulos. Privacy preservation by disassociation. PVLDB, 5(10), 2012.
43. M. Terrovitis and N. Mamoulis. Privacy Preservation in the Publication of Trajectories. In MDM, 2008.
44. M. Terrovitis, N. Mamoulis, and P. Kalnis. Privacy- preserving Anonymization of Set-valued Data. PVLDB, 1(1), 2008.
45. M. Terrovitis, N. Mamoulis, and P. Kalnis. Local and global recoding methods for anonymizing set-valued data. The VLDB Journal, 2010.
46. K. Y. X. Xiao and Y. Tao. The hardness and approximation algorithms for l-diversity. In EDBT, 2010.
47. X. Xiao, G. Bender, M. Hay, and J. Gehrke. ireduct: differential privacy with reduced relative errors. In SIGMOD, 2011.
48. X. Xiao and Y. Tao. Anatomy: simple and effective privacy preservation. In VLDB, pages 139–150, 2006.
49. J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A. Fu. Utility-Based Anonymization Using Local Recoding. In KDD, 2006.
50. Y. Xu, K. Wang, A. W.-C. Fu, and P. S. Yu. Anonymizing transaction databases for publication. In KDD, pages 767–775, 2008.
51. M. Xue, P. Karras, C. Ra¨ıssi, P. Kalnis, and H. K. Pung. Delineating social network data anonymization via random edge perturbation. In CIKM, pages 475–484, 2012.
52. R. Yarovoy, F. Bonchi, L. V. Lakshmanan, and W. H. Wang. Anonymizing moving objects: how to hide a mob in a crowd? In EDBT, 2009.
53. Q. Zhang, N. Koudas, D. Srivastava, and T. Yu. Aggregate Query Answering on Anonymized Tables.

ISSN 2349-4506
Impact Factor: 2.265

In ICDE, pages 116–125, 2007.
54. L. Zou, L. Chen, and M. T. O¨ zsu. K-automorphism: A general framework for privacy preserving network publication. PVLDB, 2(1):946–957, 2009.