A

Privacy-Preserving Publishing of Hierarchical Data

ISMET OZALP, Sabanci University
MEHMET EMRE GURSOY, University of California Los Angeles
MEHMET ERCAN NERGIZ, Zirve University
YUCEL SAYGIN, Sabanci University

Many applications today rely on storage and management of semi-structured information, e.g., XML databases and document-oriented databases. This data often has to be shared with untrusted third parties, which makes individuals' privacy a fundamental problem. In this paper, we propose anonymization techniques for privacy-preserving publishing of hierarchical data. We show that the problem of anonymizing hierarchical data poses unique challenges that cannot be readily solved by existing mechanisms. We extend two standards for privacy protection in tabular data (k-anonymity and ℓ-diversity) and apply them to hierarchical data. We present utility-aware algorithms that enforce these definitions of privacy using generalizations and suppressions of data values. To evaluate our algorithms and their heuristics, we experiment on synthetic and real datasets obtained from two universities. Our experiments show that we significantly outperform related methods that provide comparable privacy guarantees.

Categories and Subject Descriptors: H.2.7 [**Database Administration**]: Security, integrity, and protection; K.4.1 [**Public Policy Issues**]: Privacy

General Terms: Security, Algorithms, Experimentation

Additional Key Words and Phrases: Data privacy, anonymity, data publishing, k-anonymity, hierarchical data, complex data, XML.

ACM Reference Format:
Ismet Ozalp, Mehmet Emre Gursoy, Mehmet Ercan Nergiz, and Yucel Saygin, 2016. Privacy-preserving publishing of hierarchical data. *ACM* V, N, Article A (January YYYY), 28 pages.
DOI: http://dx.doi.org/10.1145/2976738

1. INTRODUCTION

The ever-increasing ability to collect and store person-specific microdata has inevitably raised concerns over individuals' privacy. Data in today's world often comes in various complex structures and formats. In particular, hierarchical data has become ubiquitous with the advent of document-oriented databases following the NoSQL trend (e.g., MongoDB) and the popularity of markup languages for richly structured documents and objects (e.g., XML, JSON, YAML). Such data contains valuable information that can be harvested through data mining techniques. However, proper de-identification and anonymization is needed before data is published, i.e., shared with untrusted third parties.

The least one can do to protect privacy is to delete explicitly identifying information (e.g., SSN, name). However, it has been shown that this is ineffective: [Sweeney

Author's addresses: I. Ozalp and Y. Saygin, Faculty of Engineering and Natural Sciences, Sabanci University; M. E. Gursoy, Department of Computer Science, University of California Los Angeles; M. E. Nergiz, Computer Engineering Department, Zirve University.

DOI: http://dx.doi.org/10.1145/2976738

2000] and [Sweeney 2002b] report that a set of *quasi-identifier (QI)* attributes (e.g., gender, zipcode, date of birth) can uniquely identify the majority of a population and also lead to *linkage attacks* [Fung et al. 2010]. An adversary performs a linkage attack by knowing one or more QI values of his victim, and trying to infer the victim's *sensitive attribute (SA)* (e.g., GPA, health condition) values.

Privacy in tabular data has been widely studied. A prominent method in data anonymization is k-anonymity, which states that each record in a k-anonymous dataset must be indistinguishable from $k-1$ other records with respect to their QIs. Such QI-wise equivalent groups are called equivalence classes (EC). k-anonymity is a promising step towards privacy, but it is still susceptible to attacks [Machanavajjhala et al. 2007; Truta and Vinay 2006]. The main concern regarding k-anonymity is that it does not consider the distribution of sensitive attributes, e.g., all individuals in an EC may have the same sensitive value. ℓ-diversity [Machanavajjhala et al. 2007] was proposed to address this problem, and requires that sensitive values in each EC are *well-represented*. To achieve this, given an EC we limit an adversary's probability of inferring a sensitive value by $1/\ell$.

Two popular ways of achieving k-anonymity and ℓ-diversity are *generalizations* and *suppressions*. Generalizations replace specific values by more general ones, e.g., course ID "*CS305*" can be replaced by "*CS 3rd year*" or "*CS3***". Suppressions conceal information by deleting it: Records that exist in the original data are completely removed from the final output. Since we are working with records with complex structures, we will not only use removal of entire records (i.e., full suppressions), but also partial suppressions (i.e., pruning data records by removing vertices, edges and subtrees). Data perturbation and the addition of counterfeits (i.e., fake information) is beyond the scope of our anonymization strategy, since we would like the data publisher to remain truthful (i.e., all data in the output must have originated from the input, and not be randomly spawned by the anonymization algorithm).

We motivate privacy-related attacks on hierarchical data using the example in Fig. 1. This record fits the hierarchical education schema given in Fig. 2. Student S, born in *1993* and majoring in *Computer Science*, took two classes: *CS201* and *CS306*. For *CS201*, S submitted evaluations for two of his instructors. For *CS306*, S submitted one evaluation and also reported that he bought the *Intro to Databases* book. We say that all of this knowledge are QIs of S. Notice that we write QIs as labels of vertices. Knowing some or all of these QIs, the goal of the adversary is to learn sensitive information about S (e.g., GPA, letter grades S received from the two classes, his evaluation scores etc.). Without anonymization this could be trivial: If there is only one Computer Science student born in 1993 in the database, then the adversary immediately learns the GPA of S (and consequently, every other sensitive value in S's data record). Our anonymization strategy is to create equivalence classes of size $\geq \ell$ for an input parameter ℓ, such that even though the adversary knows all of S's QIs, he can only link S to a group of ℓ records. Furthermore, using ℓ-diversity, we ensure that sensitive values for each vertex are well-represented, e.g., if $\ell = 3$, an EC of size 3 that contains S will have two more students that took *CS201* and they all received different letter grades. Therefore, the adversary (1) cannot distinguish S from the other two records, and (2) cannot infer with probability $> 1/\ell$ any particular sensitive value of S. In the upcoming sections we show that it is not trivial to offer this privacy guarantee. In particular, straightforward application of existing k-anonymity and ℓ-diversity algorithms are not sufficient.

Adversarial Model. We assume that adversaries have background information regarding their victims' QI values. An adversary may know any combination of QI values in the same or different vertices of his victims' records. An adversary may also exploit

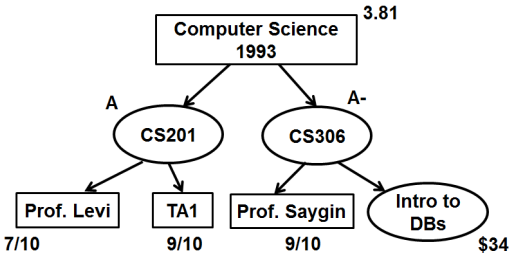


Fig. 1: A student's hierarchical data record.

structural/semantic links, e.g., S has taken 2 classes and bought exactly one book for *CS306*. Our anonymization technique therefore ensures anonymity with respect to records' structure as well as QIs. Our approach also covers negative knowledge (e.g., S did not take *CS204*) as well as positive knowledge (e.g., S took *CS201*). We assume that adversaries have no knowledge (positive or negative) of individuals' sensitive values.

Contributions. This paper makes the following contributions:

— We demonstrate the plausibility of privacy attacks on hierarchical data, e.g., XML. We show how hierarchical data anonymization differs from other data models in the literature.
— We formally define two notions of privacy, k-anonymity and ℓ-diversity, for hierarchical data. We extend popular anonymization methods (generalizations and suppressions) and utility metrics (e.g., *Information Loss Metric* LM) so that they can be applied to hierarchical data.
— We devise an anonymization algorithm that, given a collection of hierarchical data records, generates an ℓ-diverse output. We experimentally validate the usefulness of our algorithm and its heuristics.

Organization. The remainder of this paper is organized as follows: An overview of related work is given in Section 2. In Section 3, we formally define our data model and anonymization techniques, and state related assumptions. Section 4 motivates our approach by explaining why ℓ-diversity is needed and why existing tabular ℓ-diversity methods are unable to ensure ℓ-diversity in hierarchical data. Section 5 proposes a novel anonymization algorithm based on clustering, with certain heuristics. We summarize our experiments and discuss our results in Section 6. Finally, Section 7 reiterates the main points, briefly touches on future work and concludes the paper.

2. RELATED WORK

k-anonymity was proposed by Sweeney and Samarati and since then has become a standard for privacy protection [Samarati and Sweeney 1998; Sweeney 2002b]. It has been shown that optimal k-anonymity using generalizations and suppressions is NP-hard [LeFevre et al. 2006; Meyerson and Williams 2004]. Yet, achieving practical and efficient k-anonymity on tabular data has been an active area of research [Bayardo and Agrawal 2005; Iyengar 2002; LeFevre et al. 2005; Nergiz et al. 2011; Sweeney 2002a]. The main concern regarding k-anonymity is that it does not consider the distribution of sensitive values [Truta and Vinay 2006] and it is therefore susceptible to attribute linkage attacks [Fung et al. 2010]. In this paper, we use ℓ-diversity [Machanavajjhala et al. 2007] that addresses this problem. In [Xiao et al. 2010], authors show that achieving optimal ℓ-diversity through generalizations is NP-hard for $\ell \geq 3$. Among notable

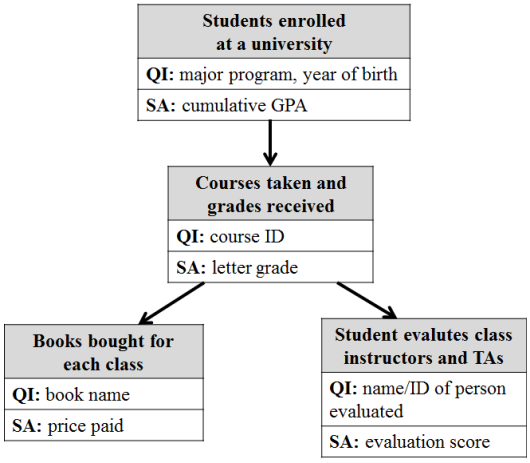


Fig. 2: Schema for education data.

ℓ-diversity algorithms are those in [Liu and Wang 2010; Machanavajjhala et al. 2007] and [Xiao et al. 2010].

Privacy notions such as k-anonymity and ℓ-diversity were initially introduced for tabular data, but they are being extended and applied to various types of complex data. Here we describe the differences between our data model and those presented in earlier works in complex data anonymization. In [Cheng et al. 2010], [Liu and Terzi 2008] and [Zheleva and Getoor 2008], authors study variations of k-anonymity (e.g., k-isomorphism) to anonymize graph data. In graph data and social network anonymization ([Zhou et al. 2008]) data often comes in the form of one large graph, and the goal is to make each vertex isomorphic or indistinguishable from $k-1$ other vertices. On the other hand, our data model assumes one disjoint record per individual. Also, we presume an explicit hierarchy between vertices, and do not allow cyclic graphs. In [Ghinita et al. 2011], [He and Naughton 2009], [Terrovitis et al. 2008] and [Terrovitis et al. 2011], authors investigate privacy-preserving publishing of transactional databases and set-valued data. Elements in set-valued data do not contain an order or a hierarchy, and all elements in a database originate from the same domain (e.g., market purchases, search logs). Our work considers multiple QI and sensitive attributes that each have a separate domain. Several studies (e.g., [Cicek et al. 2014], [Nergiz et al. 2008] and [Terrovitis and Mamoulis 2008]) use generalizations and suppressions for privacy preservation in spatio-temporal and trajectory data publishing. A trajectory is an ordered set of points where each point has one immediate neighbor (i.e, $a \rightarrow b \rightarrow c$). Whereas in hierarchical data, each vertex has multiple children that are potentially from different domains.

Several studies investigate privacy in semi-structured and hierarchical data from the point of view of access control. In particular, access control systems for XML documents have been designed and implemented for over a decade [Bertino et al. 2000; Damiani et al. 2002; Fundulaki and Marx 2004]. However, these are orthogonal to our approach: We assume that an adversary will have full knowledge over the database once it is published. In contrast, access control methods stop unauthorized users (such as adversaries) from gaining access to sensitive information in the data.

Most closely related to our work are [Gkountouna and Terrovitis 2015], [Landberg et al. 2014], [Nergiz et al. 2009] and [Yang and Li 2004] that study privacy-preserving publishing of hierarchical or tree-structured data. In [Yang and Li 2004], authors fo-

cus on cases where functional dependencies in XML data cause information leakage. They formulate such dependencies as XML constraints. They propose an algorithm that sanitizes XML documents according to these constraints so that the resulting document no longer leaks information. Our adversarial model is broader: We study adversaries that also have background knowledge regarding their victims. In [Landberg et al. 2014], authors introduce two anonymization schemes for XML data: an extension of *anatomy* [Xiao and Tao 2006a] (another well-known privacy protection method) and δ-dependency. However, these methods transform the schema of XML documents by de-associating QIs and SAs. Also, they support generalizations of SAs, which intuitively work against our goal of making records ℓ-diverse. Simultaneous to our study, [Gkountouna and Terrovitis 2015] proposed the $k^{(m,n)}$-anonymity definition for tree-structured data. In their work, attackers' background knowledge is limited to m vertex labels and n structural relations between vertices (i.e., ancestor/descendant relationships). Also contrary to our approach, they support *structural disassociations* which modify the original schema of records. In addition, they employ a global recoding approach, i.e., if a value is generalized, then all its appearances in the database must be replaced by the generalized value. This requirement can be too constraining for high-dimensional and sparse data, and therefore our solution uses local recoding that allows a value and its generalization to co-exist in the output. Furthermore, their solution is exponential in m. In [Nergiz et al. 2009], authors extend k-anonymity to anonymize multi-relational databases that have snowflake-shaped entity-relationship diagrams. Their definitions are primarily concerned with k-anonymity, and although they propose a method for ℓ-diversity, (1) their solution k-anonymizes the database first and then iteratively tries to find an output that is ℓ-diverse, and (2) they do not provide any experimental results. The effectiveness of their approach relies heavily on the k-anonymized database, which is obtained without taking SAs into account. On the other hand, our algorithm checks for ℓ-diversity at each anonymization step.

3. PROBLEM FORMULATION

3.1. Data Model

In this section we formally state our assumptions regarding the structure of our data and introduce our notation.

Definition 3.1. ***(Rooted tree)*** Let T be a graph with n vertices. We say that T is a rooted tree if and only if:

(1) T is a directed acyclic graph with $n-1$ edges.
(2) One vertex is singled out as the root vertex, and there is a single path from the root vertex to every other vertex in T.
(3) Let $children(v) = \{c_1, ..., c_m\}$ denote the children of vertex v, i.e., there exists an edge $v \rightarrow c_i$ if and only if $c_i \in children(v)$. Then, $c_1, .., c_m$ are called *siblings* of one another, and we assume no ordering among them.

We denote such trees by $T(V, E)$ where V is the set of vertices and E is the set of edges in the tree.

Definition 3.2. ***(Hierarchical data record)*** We say that a hierarchical data record satisfies the following conditions:

(1) It follows a rooted tree structure.
(2) Each vertex v has two j-tuples ($j \geq 0$) v_{QIt} and v_{QI}, where v_{QIt} contains the names of QI attributes and v_{QI} contains the values of corresponding QIs.
(3) Each vertex v also has two m-tuples ($0 \leq m \leq 1$) v_{SAt} and v_{SA}, where v_{SAt} contains the name of SA and v_{SA} contains the value of corresponding SA.

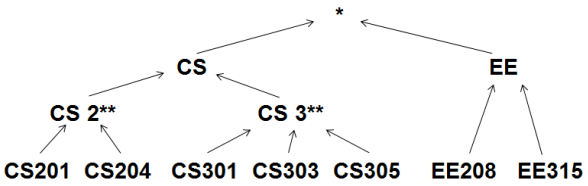


Fig. 3: Sample generalization hierarchy for course IDs.

(4) We assume $(|v_{QI}| + |v_{SA}|) \geq 1$ to eliminate empty vertices.

In our examples we adopt the following notation to represent hierarchical records: We write QI values (v_{QI}) as labels of tree vertices and associated SA values (v_{SA}) right next to the vertices (as contiguous information). For the root vertex in Fig. 1, v_{QIt}*=(major program, year of birth)*, v_{SAt}*=(GPA)*, v_{QI}*=(Computer Science, 1993)* and v_{SA}*=(3.81)*. An edge between two vertices signals that information is semantically linked, e.g., the evaluation score of *9/10* for *Prof. Saygin* in Fig. 1 was given by this particular student and for the *CS306* class. Such links can be established through primary and foreign keys in a multi-relational SQL database, or through hierarchical object representations in XML or JSON. Conversion of any type of hierarchical data to the structure defined above is trivial, given which attributes are quasi-identifiers and which ones are sensitive.

We say that an individual's record in the database conforms to the definition of a hierarchical data record, and only one hierarchical record exists per individual. The database is a collection F that contains n hierarchical records, denoted $T_1, ..., T_n$.

Let $v_X[i]$ denote the i'th element in the r-tuple v_X, where $r = j$ or m. Let $\Omega(A)$ denote the domain of attribute A. We assume, without loss of generality, that the domains of different attributes are mutually exclusive: $\Omega(A) \cap \Omega(A^{'}) = \emptyset$ for $A \neq A^{'}$. We also require: $\forall i \in \{1, .., |v_{QI}|\}$, $v_{QI}[i] \in \Omega(v_{QIt}[i])$. Likewise, if the vertex contains a sensitive attribute (i.e., $|v_{SA}| = 1$), then $v_{SA}[1] \in \Omega(v_{SAt}[1])$.

Definition 3.3. ***(Union-compatibility)*** Two vertices v and $v^{'}$ are union-compatible if and only if $v_{QIt} = v^{'}_{QIt}$ and $v_{SAt} = v^{'}_{SAt}$.

We use union-compatibility akin to database relations: Two database relations are union-compatible if they share the same number of attributes and each attribute is from the same domain. Similarly, in our case, two vertices are union-compatible if they follow the same schema (i.e., same QIs and SAs).

3.2. Anonymization

Domain generalization hierarchies (DGH) are taxonomy trees that provide a hierarchical order and categorization of values. We assume that a DGH is either available or easily inferable for each QI. Note that this assumption is widely adopted in the anonymization literature [Fung et al. 2010; Nergiz et al. 2009]. Values observed in the database appear as the leaves of DGHs. The root vertices of DGHs contain "*" to mean "any value", i.e., value completely hidden. A DGH is given for attribute *course ID* in Fig. 3.

Definition 3.4. ***(Generalization function)*** For two data values x and x^* from the same QI attribute A, x^* is a valid generalization of x, written $x^* \in \phi(x)$, if and only if x^* appears as an ancestor of x in the DGH of A. We abuse notation and write $\phi_l^{-1}(x^*)$

to indicate all possible leaves that can be generalized to value x^* using valid generalizations.

For example, for the QI *course ID*, *CS3*** $\in \phi$(*CS303*) and *CS* $\in \phi$(*CS303*), whereas *CS2*** $\notin \phi$(*CS303*). Also, ϕ_l^{-1}(*CS3***) = {*CS301, CS303, CS305*}, and ϕ_l^{-1}(*CS305*) = {*CS305*}.

Definition 3.5. ***(Vertex generalization)*** We say that vertex v^* is a valid generalization of v and write $v^* \in \Delta(v)$, if:

(1) v and v^* are union-compatible.
(2) $v_{QI} \neq v^*_{QI}$.
(3) $\forall a^* \in v^*_{QI}$, either $a^* \in v_{QI}$ or there exists $a \in v_{QI}$ such that $a^* \in \phi(a)$.
(4) $v_{SA} = v^*_{SA}$.

In words, a vertex is generalized when at least one of its QI values gets replaced by a value that is more general according to the attribute's DGH. A vertex generalization leaves sensitive values intact.

In tabular data, suppression of a row refers to the removal of that row from the published dataset (or equivalently, all values in that row are replaced by "*"). In our setting, this translates to completely removing an individual's hierarchical record. Although this might be necessary and we support this operation, its effect is also drastic: If the deleted record is large (i.e., contains a lot of vertices), then a lot of useful information might be lost. We therefore introduce partial suppressions.

Definition 3.6. ***(Partial suppression)*** We say that a hierarchical data record T^* is a partially suppressed version of T, if T^* is obtained from T by first removing exactly one edge from T (call this e) and then deleting all vertices and edges that are no longer accessible from the root of T (i.e., there is no longer a path from the root to them). We write $T^* = \varphi_e(T)$ to denote this operation.

Intuitively, a partial suppression is nothing but tree pruning. Such pruning can lead to the deletion of a single vertex or a subtree containing multiple vertices and edges. Note that the remainder of the data record is untouched, i.e., vertices that "survive" the partial suppression operation incur no changes to their QIs or sensitive values. Fig. 4 contains several examples: From Fig. 4a to Fig. 4b, the upper record loses the vertex with *TA5* under *CS404*. From Fig. 4a to Fig. 4c, the edge between the root and *CS404* is broken, which leads to the suppression of a larger subtree (i.e., children of *CS404* are also deleted). We explicitly replace suppressed vertices with dashed lines and lost information (both v_{QI} and v_{SA}) with "*" for demonstration purposes. They are otherwise not part of the output.

Definition 3.7. ***(ℓ-diversity)*** Let $X = \{s_1, s_2, ..., s_n\}$ be a multiset of values from the domain of a sensitive attribute A, i.e., $s_i \in \Omega(A)$. Let $f(s_i)$ denote the frequency of value s_i in X. Then, X is ℓ-diverse if for all s_i, $f(s_i) \leq 1/\ell$.

Informally, this probabilistic ℓ-diversity definition states that the frequency of all sensitive values must be bounded by $1/\ell$.

Sensitive attributes can be categorical (e.g., letter grade) or continuous (e.g., GPA). The domain of categorical SAs consists of discrete values (e.g., letter grades from A to F), and it is straightforward to evaluate ℓ-diversity on a set of discrete values as above. However, continuous SAs require an intermediate discretization step. The domain of a continuous SA is divided into non-overlapping buckets, and X then contains the buckets data values fall into. (E.g., GPA domain $[0.0 - 4.0]$ can be divided into 8 buckets of size 0.5. A GPA value 3.26 can then translate to the bucket $[3.0 - 3.50)$.) We do not

enforce a specific discretization, instead our algorithms can work with an arbitrary discretization that meets the demands and preferences of the data publisher. We also allow discretizations to contain buckets with different sizes.

Definition 3.8. ***(Diversity of vertices)*** Let $V = \{v^1, ..., v^n\}$ be a set of vertices from hierarchical data records. We study two cases:

— For $v^j \in V$, $|v^j_{SA}| = 0$. Then, V is ℓ-diverse if all vertices in V are pairwise union-compatible.
— For $v^j \in V$, $|v^j_{SA}| = 1$. Let X be defined as $X = \{v^1_{SA}[1], v^2_{SA}[1], ..., v^n_{SA}[1]\}$. Then, V is ℓ-diverse if all vertices in V are pairwise union-compatible and X is ℓ-diverse.

Various metrics were proposed and used in relevant literature to calculate costs of anonymization [Bayardo and Agrawal 2005; Bertino et al. 2008; Iyengar 2002; Xiao and Tao 2006b]. In this paper, we will use an extension of the general *loss metric* (LM) [Nergiz et al. 2011]. Similar extensions were previously applied in a number of settings, including medical health records [Tamersoy et al. 2012] and multi-relational databases [Nergiz et al. 2009].

Definition 3.9. ***(Individual LM cost)*** Given a DGH for attribute A and a value $x \in \Omega(A)$ (i.e., x exists in A's DGH), the individual LM cost of value x is:

$$LM'(x) = \frac{|\phi_l^{-1}(x)| - 1}{|\phi_l^{-1}(r)| - 1}$$

where r denotes the root of A's DGH.

Definition 3.10. ***(LM cost of a collection of hierarchical records)*** Let F and F^* be collections of hierarchical data records, where F^* is obtained via anonymizing F. Let Ψ denote the set of all vertices that exist in F but do not exist in F^* due to partial or full suppressions of records. Then, the LM cost of F^* is:

$$LM(F^*) = \frac{(\sum_{T_i^* \in F^*} \sum_{v^* \in T_i^*} \sum_{q^* \in v^*_{QI}} LM'(q^*)) + (\sum_{p \in \Psi} |p_{QI}|)}{\sum_{T_i \in F} \sum_{v \in T_i} |v_{QI}|}$$

For example, according to Fig. 3, $LM'(CS) = 4/6$ and $LM'(CS2^{**}) = 1/6$. We use LM', defined on a single QI value, to build a metric suitable to our setting. In this new definition, the sum is broken down into two factors: The first factor calculates the cost incurred by generalizations of vertices that appear in the published data. The second factor adds the cost of suppressions. The total cost is calculated on the order of labels rather than vertices or trees, to better focus on each individual piece of data lost during anonymization.

One can verify that the LM' cost of a label is within the range [0, 1], where the root of a DGH receives the highest penalty (1) and leaves receive no penalty (0). Consequently, we ensure that $LM(F^*)$ is also normalized to a value within [0, 1].

We compute the LM cost of anonymizing the two records in Fig. 4c to provide an example for $LM(F^*)$. Assume that F consists of only the two records in Fig. 4a, and F^* is the records in Fig. 4c. Further assume the LM costs of generalizing years of birth *1994* and *1995* to *199** is $1/10$, course IDs *CS306* and *CS305* to *CS3*** is $1/3$, instructors *Prof. Saygin* and *Prof. Nergiz* to *DB Prof.* is $2/7$, and *TA1* and *TA2* to *TA* is $1/2$. Then,

$$LM(F^*) = \frac{(\frac{1}{10} + \frac{1}{3} + \frac{2}{7} + \frac{1}{2}) \cdot 2 + 7}{19} = 0.497$$

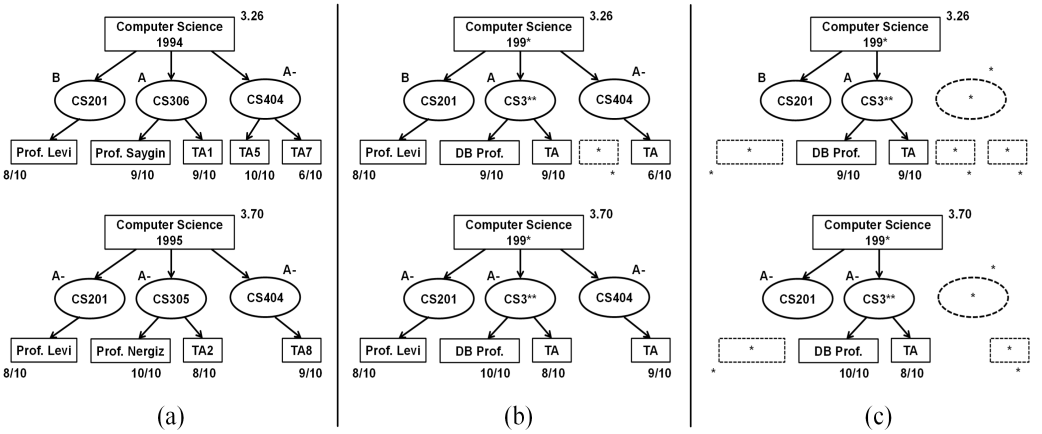


Fig. 4: (a) Hierarchical data records for two sample students. (b) A 2-anonymous version of these records. (c) A 2-diverse version of these records.

3.3. Problem Definition

Having established the preliminaries, in this section we formally define and state the problem. We start with an auxiliary definition regarding QI-isomorphism.

Definition 3.11. ***(QI-isomorphism)*** Let $T_1(V_1, E_1)$ denote a hierarchical data record with a set of vertices V_1 and edges E_1. A data record $T_2(V_2, E_2)$ is QI-isomorphic to T_1 if and only if there exists a bijection $f : V_1 \rightarrow V_2$ such that:

(1) There exists an edge $e_i \in E_2$ from $f(x)$ to $f(y)$ if and only if there exists an edge $e_j \in E_1$ from x to y.
(2) The root vertex is conserved; i.e., denoting the root of the first tree as $r_1 \in V_1$ and the root of the second tree as $r_2 \in V_2$, $f(r_1) = r_2$.
(3) For all pairs (x, x'), where $x \in V_1$ and $x' = f(x)$, x and x' are union-compatible and $x_{QI} = x'_{QI}$.

Definition 3.12. ***(Equivalence class of hierarchical records)*** We say that records $D = \{T_1, .., T_k\}$ are k-anonymous and form an equivalence class, if all possible pairs (T_i, T_j), where $T_i \in D$ and $T_j \in D$, are QI-isomorphic.

Two records are QI-isomorphic if they appear to be completely same when all sensitive values are deleted from both. In other words, they are indistinguishable in terms of labels and structure. There is a clear analogy between the traditional definition of equivalence classes in tabular k-anonymity and our definition for hierarchical records: Both state that an equivalence class is a set of records that are indistinguishable with respect to their QIs.

Definition 3.13. ***(ℓ-diverse equivalence class)*** We say that records $\{T_1, .., T_k\}$ form an ℓ-diverse equivalence class, if and only if:

(1) $\{T_1, .., T_k\}$ constitute an equivalence class.
(2) For $1 \leq i \leq k-1$, let f_i be a bijection that maps T_1's vertices to T_{i+1}'s vertices, as in QI-isomorphism. Let T_1 have n vertices, labeled arbitrarily as $v_1^1, v_2^1, v_3^1, .., v_n^1$. Then, there should exist a set of bijections $\{f_1, f_2, .., f_{k-1}\}$ such that $\forall x \in \{1, 2, .., n\}$, the set of vertices $V = \{v_x^1, f_1(v_x^1), f_2(v_x^1), .., f_{k-1}(v_x^1)\}$ is ℓ-diverse.

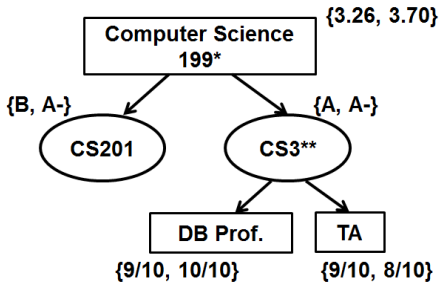


Fig. 5: A class representative.

Fig. 4 contains two records together with their 2-anonymous and 2-diverse versions. This is just one way of anonymizing these records, there are also other correct (i.e., fitting the definition of *anonymity* and *diversity*) anonymizations. The quality of these anonymizations, however, depend on how much information is lost (according to an appropriate cost metric). An anonymization that satisfies k-anonymity or ℓ-diversity and yields the lowest information loss is most desirable.

We should point out that multiple bijections between two records' vertices are possible if they contain multiple union-compatible sibling vertices with identical QIs. In such cases, it is too restrictive to require that all possible bijections satisfy ℓ-diversity, therefore our definition states that it would suffice to have one bijection that does.

An alternative representation of an equivalence class which we use in later sections is the class representative for a given equivalence class. A class representative $\widehat{T}$ is essentially a hierarchical data record with one extension: If a vertex contains a sensitive attribute, its value is not a single element, but rather a list of elements. ($\forall v \in \widehat{T}$, v_{SA} returns a set rather than a single sensitive value.) We formally define class representative as follows:

Definition 3.14. ***(Class representative)*** Given an equivalence class $D = \{T_1, .., T_k\}$ with the corresponding set of bijections $\{f_1, f_2, .., f_{k-1}\}$, we say $\widehat{T}$ is the class representative for D if $\widehat{T}$ is QI-isomorphic to T_1 with a bijection function f and $\forall v \in \widehat{T}$, $v_{SA} = \{f(v)_{SA}, f_1(f(v))_{SA}, \ldots, f_{k-1}(f(v))_{SA}\}$.

Fig. 5 shows a representative for the equivalence class given in Fig. 4c. It is easy to show that a given equivalence class is ℓ-diverse if and only if the corresponding representative is ℓ-diverse, that is $\forall v \in \widehat{T}$, the set v_{SA} satisfies ℓ-diversity.

Definition 3.15. ***(ℓ-diversity of a database)*** A collection of records $F^*(T_1^*, ..., T_n^*)$ is ℓ-diverse if all records $T_i^* \in F^*$ belong to exactly one ℓ_j-diverse equivalence class, and for all ℓ_j, $\ell_j \geq \ell$ holds.

Definition 3.16. ***(Anonymization of a database)*** Given a collection of hierarchical data records $F(T_1, T_2, ..., T_n)$, an anonymized output F^* is generated via the following principle: For each record $T_i \in F$, either T_i is fully suppressed and does not appear in F^*, or T_i is transformed into $T_i^* \in F^*$ by performing a set of generalizations $\{\Delta\}$ and partial suppressions $\{\varphi_e(T_i)\}$.

With these definitions in mind, the problem we study in this paper can be stated as follows: Given a set of hierarchical records F, we would like to compute an ℓ-diverse output F^* with minimal information loss, using the anonymization principle above.

Students enrolled at the university			
studentID	major	YoB	GPA
S1	Computer Science	1992	3.25
S2	Computer Science	1993	2.67

Courses and letter grades		
studentID	class	grade
S1	CS201	A
S1	CS202	A-
S2	CS401	B
S2	CS404	B+

Fig. 6: Students *S1* and *S2* and their classes as two tables linked using studentIDs (primary key in Table 1, foreign key in Table 2).

Students enrolled at the university			
studentID	major	YoB	GPA
S1	Computer Science	199*	3.25
S2	Computer Science	199*	2.67

Courses and letter grades		
studentID	class	grade
S1	CS2**	A
S1	CS2**	A-
S2	CS4**	B
S2	CS4**	B+

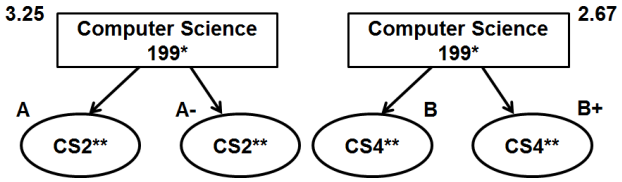


Fig. 7: Potential result if the two tables in Fig. 6 are anonymized independently.

4. MOTIVATION

Given our definitions of privacy, two natural questions to ask are "why ℓ-diversity" and "why does one need new algorithms to enforce ℓ-diversity on hierarchical data". In this section we aim to provide answers to these two questions.

4.1. ℓ-diversity vs k-anonymity in hierarchical data

Prior approaches in hierarchical (and tree-structured) data anonymization against linkage attacks can be divided into two camps: providing privacy by disassociating QIs and SAs [Landberg et al. 2014] and extensions of k-anonymity (e.g., multi-relational k-anonymity [Nergiz et al. 2009] and $k^{(m,n)}$-anonymity [Gkountouna and Terrovitis 2015]). The former publishes QI values and SA values separately, hence an adversary cannot determine the sensitive value of a particular vertex (e.g., the letter grade S received from class *CS201*). In the latter, records are anonymized in terms of structure and labels (QIs in our case), but sensitive values are left unattended. (In particular, [Gkountouna and Terrovitis 2015] has no distinction between QI and SA.) Both may result in equivalence classes that leak sensitive values with significant probabilities.

Let us demonstrate the plausibility of homogeneity and background knowledge attacks on hierarchical data, where data is k-anonymized according to [Nergiz et al. 2009] or [Gkountouna and Terrovitis 2015]. Say that a 2-anonymous dataset has been published, such as the one in Fig. 4b. Let the adversary know beforehand that there will be at most two students that majored in *Computer Science* and were born in the *1990s*. His victim, S is among these two students. The adversary links S to the records in Fig. 4b. At this point, the published dataset leaks the following pieces of information: (1) S received an *A-* from *CS404*. (2) S submitted an evaluation score of *8* for *Prof. Levi* in *CS201*. The peculiarity of this example comes from the fact that the ad-

Join key	*Quasi-Identifiers*			*Sensitive Attributes*	
studentID	**major**	**YoB**	**class**	**grade**	**GPA**
S3	Computer Science	1993	CS301	C+	2.17
S4	Computer Science	1995	CS305	A	3.78
S4	Computer Science	1995	CS306	A-	3.78
S5	Computer Science	1994	CS301	B	2.96

Fig. 8: Universal relation constructed by joining the *Enrollment* and *Courses* relations with students *S3*, *S4* and *S5* using studentIDs.

versary had no knowledge of QI values for the vertices that leaked these information (e.g., the adversary did not know that S evaluated *Prof. Levi*). Both of these privacy leaks could have been avoided if the published data was 2-diverse as in Fig. 4c.

4.2. ℓ-diversity in tabular vs hierarchical data

As reported earlier, several algorithms that apply ℓ-diversity to tabular data have been implemented. In applicable situations, one way of processing hierarchical data is to reduce it to tabular data and then run tabular algorithms on it. There are also arguments that say in most scenarios, converting hierarchical data to a single giant relation and then using single-table algorithms is undesirable because of potential loss of information and semantic links between data records [Han et al. 2011]. We now demonstrate that such conversions and reductions are not sufficient also for privacy protection.

4.2.1. Anonymizing relations separately. A hierarchical schema (e.g., Fig. 2) can be represented using multiple database relations that are linked via primary and foreign keys (i.e., join keys). Then, a straightforward approach would be to consider each relation independently and run tabular ℓ-diversity algorithms on them.

Consider the two tables in Fig. 6, where *studentID*s are added and used as join keys. When these two tables are treated independently, a resulting anonymization could be the one in Fig. 7. It can easily be verified that both tables are 2-diverse by themselves. Converting the result into our hierarchical representation, though, we see that students *S1* and *S2* are neither 2-anonymous nor 2-diverse. An adversary that knows *S1* took *CS201* learns the GPA of *S1*, since *S2* has not taken any *CS200*-series classes.

The main problem of this *independent anonymization* approach is that anonymizations are not guaranteed to be consistent between multiple tables. In the first table, *S1* and *S2*'s tuples are anonymized with respect to each other, but a tabular anonymization algorithm does not acknowledge this when anonymizing the second table. Hence, *S1*'s tuples may be bundled together and *S2*'s tuples may be bundled together while creating a 2-diverse version of the second table.

4.2.2. Constructing and anonymizing a universal relation. Another approach is to *flatten* hierarchical data into one big relation called the *universal relation*, i.e., the universal relation is obtained by joining all relations in a hierarchical schema using join keys. Fig. 8 provides a sample universal relation. Notice that this creates a significant amount of redundancy and undesirable dependencies. Information in deeper vertices of the records have to be rewritten for each descendant connected to that vertex (e.g., QIs major and year of birth are repeated for each class taken). A second problem is that leaf vertices in a data record may be at different depths, which will force work-arounds such as having *null* values in the universal relation. Here we show the ineffectiveness of the *universal relation* approach even ignoring the problems discussed up to this point.

Join key	*Quasi-Identifiers*			*Sensitive Attributes*	
studentID	**major**	**YoB**	**class**	**grade**	**GPA**
S3	Computer Science	199*	CS3**	C+	2.17
S4	Computer Science	199*	CS3**	A	3.78
S4	Computer Science	199*	CS3**	A-	3.78
S5	Computer Science	199*	CS3**	B	2.96

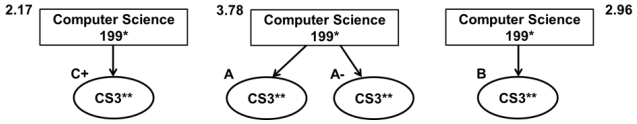


Fig. 9: 2-diverse version of the universal relation in Fig. 8.

The table in Fig. 9 is 2-diverse in terms of the two sensitive attributes, *GPA* and *grade*. However, the hierarchical records of *S3*, *S4* and *S5* are not anonymous: *S3* and *S5* are shown having taken one *CS3*** class each, but *S4* has taken two. An adversary that knows *S4* is the only student who has enrolled in more than one *CS3*** class can learn the grades *S4* received from these classes, together with *S4*'s GPA. The problem this time arises from the fact that each individual may have an unknown number of entries in the universal relation.

5. ANONYMIZATION ALGORITHM

We designed and implemented a solution to the anonymization problem stated at the end of Section 3. Before moving forward, we would like to underline two important characteristics of our anonymization scheme. First, our approach ensures that the data publisher remains truthful. The output does not contain any information that did not exist originally in the input, i.e., we do not consider adding new vertices, changing QIs of vertices (other than generalizing them), or adding new QIs or SAs to existing vertices. Second, vertices that appear in the output have the same depth, adjacency and parent as they did in the input. That is, the structure of records in the output are consistent with the input. This schema preservation enables easier data mining without any ambiguity.

We present our algorithm in two steps: (1) Given two records, we focus on how to anonymize them with respect to each other so that they become 2-diverse with low information loss. (2) We build a clustering algorithm that employs the previous step and class representatives to anonymize an arbitrary number of records.

5.1. Pairwise Anonymization

Converting two records to a 2-diverse pair is pivotal not only because we use it as a building block in our clustering algorithm, but also we employ it as a similarity metric (i.e., to calculate *distance* between two hierarchical data records). In addition, given a fixed pair of records as inputs, the anonymization function should be able to produce a 2-diverse output with as little information loss as possible. Therefore, it relies on finding vertices and subtrees that are *similar* in both records.

We define the following notation: Let $root(T)$ denote the root vertex of the hierarchical data record T, and $subtrees(v)$ denote the subtrees rooted at the children of v (i.e., for each child c_i of v, the hierarchical data record rooted at c_i is included in $subtrees(v)$). Given two QI values X and Y both from the same QI domain, and Z that is the DGH of the QI, we say that function $mrca(X, Y, Z)$ returns the lowest (i.e., most recent) common ancestor of X and Y according to Z. Assume that the function $cost(T)$ returns

the cost of anonymization of T, given a pre-defined cost metric CM. An applicable cost metric is LM, and in that case, cost of a record T is:

$$cost(T) = (\sum_{v \in V} \sum_{q \in v_{QI}} LM'(q)) + (\sum_{w \in \Psi} |w_{QI}|)$$

where V denotes the vertices in T that are not suppressed and Ψ denotes the vertices that were in T but are now suppressed. Let $clone(T)$ return a copy of T. Furthermore, given two vertices a and b, let $u\text{-}comp(a, b)$ test the union-compatibility of a and b, and $diverse(a, b)$ have the following behavior:

$$diverse(a, b) = \begin{cases} \text{true} & \text{if } \textit{u-comp(a,b)} \text{ and } a_{SA} \cap b_{SA} = \emptyset \\ \text{false} & \text{otherwise} \end{cases}$$

Algorithm 1 Top-down anonymization of hierarchical records

Input: Two hierarchical data records (or class representatives) T_1 and T_2,
anonymization cost metric for *cost* calculation,
DGHs of QI attributes for finding *mrca*
Require: |children(root(T_1))| $\leq$ |children(root(T_2))|, otherwise swap T_1 and T_2

1: **procedure** DIVERSIFY
2: $a \leftarrow$root(T_1)
3: $b \leftarrow$root(T_2)
4: **if** ¬diverse(a,b) **then**
5: suppress T_1 and T_2
6: **return** cost(T_1) + cost(T_2)
7: **for** $i = 1$ to $|a_{QI}|$ **do**
8: g $\leftarrow$ mrca($a_{QI}[i]$, $b_{QI}[i]$, DGH of $a_{QIt}[i]$)
9: replace $a_{QI}[i]$ with g
10: replace $b_{QI}[i]$ with g
11: **if** subtrees(a) $= \emptyset$ **and** subtrees(b) $= \emptyset$ **then**
12: **return** cost(T_1) + cost(T_2)
13: **else if** subtrees(a) $= \emptyset$ **and** subtrees(b) $\neq \emptyset$ **then**
14: let E be the set of outgoing edges from b
15: **for** $e \in E$ **do**
16: $T_2 \leftarrow \varphi_e(T_2)$
17: **return** cost(T_1) + cost(T_2)
18: $P \leftarrow$ FindMapping(subtrees(a), subtrees(b))
19: **for** each pair $(a_i, b_j) \in P$ **do**
20: diversify(a_i, b_j)
21: **for** $v \in$ subtrees(b) **and** $\nexists (x, v) \in P$ for some x **do**
22: Let e be the edge from b to v
23: $T_2 \leftarrow \varphi_e(T_2)$
24: **return** cost(T_1) + cost(T_2)

A function that anonymizes hierarchical records in top-down manner is presented in Algorithm 1. We refer to this function as **diversify**. Without loss of generality, we assume that for the two input hierarchical records T_1 and T_2 (rooted at a and b, respectively), |children(a)| $\leq$ |children(b)|. (Otherwise T_1 and T_2 can be interchanged as the

first step.) The algorithm can be studied in several steps. First step checks the union compatibility and diversity of root vertices a and b. If a and b cannot be anonymized, then their trees are suppressed. In the second step (lines 7-10), we generalize the QIs of a and b according to their DGHs. Resulting a and b will be indistinguishable in terms of QIs. In step 3 (lines 11-17), the algorithm checks if further calculation is needed: If a and b both have children, then we need to find a low-cost anonymization of their subtrees. If one does not have any children, then we can safely suppress the children and subtrees of the other. (Otherwise it would be impossible to achieve QI-isomorphism due to structural difference.) When the algorithm reaches line 18, it has dealt with the current level (i.e., checked if root vertices are diverse, anonymized them and ensured that both have children). A low cost pairing (i.e., mapping) between the subtrees rooted at a's children and the subtrees rooted at b's children is returned by the function *FindMapping*. (We will give a detailed explanation of how the mapping is computed in the next section.) Pairs returned by the function are suitable candidates to be anonymized with one another. Hence, *diversify* is run recursively on each pair (lines 19-20). Since we assumed $|\text{children}(a)| \leq |\text{children}(b)|$, all subtrees rooted at a's children will be paired, but some subtrees rooted at b's children might be left-overs (i.e., they remain unpaired). Unpaired subtrees are suppressed (lines 21-23) to achieve QI-isomorphism of T_1 and T_2. Finally, a successful execution of *diversify* always returns the cost of anonymizing its inputs (see the return statements throughout).

5.2. Finding a Good Mapping

Recall that *FindMapping* is called using two lists of hierarchical data records S and U (where $|S| \leq |U|$), and the goal is to produce a set of pairs $\{(s, u) \mid s \in S, u \in U\}$ that are similar. We measure similarity as the cost of anonymization. Finding an optimal solution to this problem requires finding all mappings between all elements in S and U, and picking the mapping that yields the lowest cost. However, this is infeasible: Let S have n elements and U have m elements, where $m \geq n$. The number of possible pairings between S and U is $\binom{m}{n} \cdot n!$, which implies exponential complexity. This becomes a significant problem when the branching factor of input data records is large. (Even for toy datasets with average branching factors of 6-7, optimal search took several hours.) We therefore need heuristic strategies for *FindMapping*. Based on this observation, we now describe two different solutions to the problem: one that employs a greedy algorithm, and another that models the problem as an optimization problem using linear programming.

The greedy algorithm. This heuristic traverses S by picking one element at a time, and finds the most suitable candidate in U to pair the element with. A more formal description is given in Algorithm 2. The greedy solution has no guarantees of finding a global optimum, but instead settles for a local optimum in each iteration (i.e., for each element in S).

The procedure in Algorithm 2 works as follows: We pick one record at a time from the first set S and call this record f (line 3). Then, we consider each unpaired element v in the second set U and compute the information loss of anonymizing f with v (lines 6-10). This is done by first making copies of f and v (to make explicit that we do not modify the original records) and then running *diversify* on them. The record that yields the lowest cost wins and gets to pair up with f (lines 10-14). We repeat this procedure until S is exhausted.

Reduction to an assignment problem. We propose a second strategy for *FindMapping*: We model the problem in hand as a linear sum assignment problem (LSAP). LSAP is a famous linear programming and optimization problem [Munkres 1957], where one has n agents that need to be assigned to n tasks. Assigning an agent to

Algorithm 2 Finding a low-cost mapping greedily

Input: Two lists of hierarchical data records S and U, where $|S| \leq |U|$

```
1:  procedure FINDMAPPING-GRD
2:      R ← ∅
3:      for each f ∈ S do
4:          minCost ← +∞
5:          match ← ∅
6:          for each v ∈ U do
7:              f* ← clone(f)
8:              v* ← clone(v)
9:              c ← diversify(f*, v*)
10:             if c < minCost then
11:                 minCost ← c
12:                 match ← v
13:         R ← R ∪ (f,match)
14:         U ← U − match
15:     return R
```

a task has a certain cost that depends on the task and the agent performing it. The goal is to find an assignment such that all tasks are performed by assigning one agent to each task, one task to each agent and the total cost of the assignment (i.e., linear sum of task-agent pairs selected) is minimized.

More formally, given an $n \times n$ cost matrix $C = (c_{ij})$ and a binary variable x_{ij} representing the assignment of agent i to task j, a LSAP can be modeled as:

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_{ij}$$

$$\begin{aligned}
\text{Subject to:} \quad & \sum_{i=1}^{n} x_{ij} = 1 \quad \text{for} \;\; j = 1, 2, .., n \\
& \sum_{j=1}^{n} x_{ij} = 1 \quad \text{for} \;\; i = 1, 2, .., n \\
& x_{ij} \in \{0, 1\} \quad \text{for} \;\; i, j = 1, 2, .., n \\
& c_{ij} \geq 0 \quad\quad\;\; \text{for} \;\; i, j = 1, 2, .., n
\end{aligned}$$

We use the Hungarian algorithm [Kuhn 1955] to solve a LSAP, which finds an optimal (i.e., lowest-cost) solution to the problem above in $\mathcal{O}(n^3)$ time. The solution is a collection of x_{ij}s that tell which agent is assigned to which task.

We now explain how we use LSAPs in *FindMapping*. The process is shown in Algorithm 3. Given two lists of records S and U, we treat the records in S as agents, and the records in U as tasks in a LSAP. We calculate the cost of an agent-task pair by running *diversify* on them, which computes the information loss incurred for anonymizing that pair (lines 3-7). This fills the uppermost $|S|$ rows of the cost matrix with non-negative numbers. In many cases we have $|S| < |U|$ (i.e., number of agents and tasks differ) and hence the LSAP is unbalanced [Pentico 2007]. In these cases we add dummy suppression agents to the cost matrix (lowermost $|U| - |S|$ rows) to mark unmatched elements in $|U|$ which will eventually be suppressed by *diversify*. We capture the costs of suppressing elements in U on lines 8-10. The *cost* method in Section 5.1 can be used for this, and in that case, the cost of suppressing a subtree is equal to the total number

Algorithm 3 Finding a low-cost mapping using a LSAP

Input: Two lists of hierarchical data records S and U, where $|S| \leq |U|$

```
1:  procedure FINDMAPPING-LSAP
2:      Initialize |U| × |U| cost matrix C
3:      for i = 1 to |S| do
4:          for j = 1 to |U| do
5:              f* ← clone(S[i])
6:              v* ← clone(U[j])
7:              c_ij ← diversify(f*, v*)
8:      for i = |S| + 1 to |U| do
9:          for j = 1 to |U| do
10:             c_ij ← cost of suppressing U[j]
11:     X ← solve the LSAP with cost matrix C
12:     R ← ∅
13:     for each x_ij ∈ X do
14:         if x_ij = 1 and i ≤ |S| then
15:             R ← R ∪ (S[i], U[j])
16:     return R
```

of data values (i.e., QIs) that are deleted from that subtree. On line 11 we solve the LSAP using the Hungarian algorithm, and consequently use this solution to compute the matching pairs of records in S and U that should be returned by *FindMapping*, while removing all dummy assignments (lines 13-15).

5.3. Clustering

Let c denote a cluster. Each cluster contains:

— A class representative, denoted c_{rep}. This is a summary data structure that depicts the current state of the cluster. A formal definition of class representatives was given in Definition 3.14.
— A set of data records, denoted c_{init}, that are the original (i.e., unmodified) versions of the records in the cluster.

We first explain how we initialize and build one cluster. The procedure for this is given in Algorithm 4. Essentially, we treat a cluster as an equivalence class, and use a clustering algorithm to build ℓ-diverse equivalence classes. A cluster is initialized using one record (line 3). At that point, the cluster is a 1-diverse equivalence class. In order to satisfy ℓ-diversity, this cluster needs to recruit $\ell - 1$ other records. This is achieved by finding the record T_b in F that is closest to c, i.e., we need to find T_b such that diversify(T_b, c_{rep}) would return the lowest cost (line 5). Once this record is found, it is removed from F and added to the cluster (lines 6-7). Then, c_{rep} is updated on line 8: Generalizations and suppressions are performed, and sensitive values in T_b's vertices are added to matching vertices in c_{rep}. In short, the main loop (lines 4-8) iteratively adds new records to a cluster one by one, and each record joining a t-diverse cluster makes it $(t+1)$-diverse. The process is terminated when the cluster becomes ℓ-diverse.

Based on this, we present our full clustering procedure in Algorithm 5 and refer to it as **ClusTree**. It receives a database of hierarchical data records F, a privacy parameter ℓ and two clustering parameters m and s that are both positive floating numbers. We pick a T_a from the input F (line 5) and use it to create a new cluster using *CreateCluster* (line 9). Once a cluster is formed, we calculate its total anonymization

Algorithm 4 Create ℓ-diverse cluster

Input: A collection of n data records $F(T_1, ..., T_n)$,
parameter ℓ for ℓ-diversity,
data record $T_a \in F$

procedure CREATECLUSTER
  $F \leftarrow F - T_a$
  Initialize c, where $c_{rep} = T_a$ and $c_{init} = \{T_a\}$
  for $i = 2$ to ℓ **do**
    Find $T_b \in F$ s.t. argmin_{T_b}(diversify(c_{rep}, T_b))
    $F \leftarrow F - T_b$
    $c_{init} \leftarrow c_{init} \cup T_b$
    Update c_{rep} by diversify(clone(T_b), c_{rep}) and copying sensitive values of matching vertices
  return c

Algorithm 5 Clustering algorithm

Input: A collection of n data records $F(T_1, ..., T_n)$,
parameter ℓ for ℓ-diversity,
maximum standard deviation multiplier parameter m,
step size parameter s

procedure CLUSTREE
  $R \leftarrow \emptyset$
  $\chi \leftarrow 0$
  while true **do**
    for each $T_a \in F$ **do**
      if $|F| < \ell$ **then**
        suppress all $T_i \in F$
        return R
      $c \leftarrow$ CreateCluster(F, ℓ, T_a)
      clcost $\leftarrow 0$
      for each $T_j \in c$ **do**
        clcost $\leftarrow$ clcost + cost(T_j)
      if $|R| > 1$ **then**
        Let μ be the mean of costs of clusters in R
        Let σ be the standard deviation of costs of clusters in R
        if clcost $> (\mu + \sigma * \chi)$ **then**
          Add all $T_i \in c_{init}$ back to F
          Discard cluster c
        else
          $R \leftarrow R \cup c$
      else
        $R \leftarrow R \cup c$
      if $|F| = 0$ **then**
        return R
    $\chi \leftarrow \chi + s$
    if $\chi \geq m$ **then**
      $\chi \leftarrow +\infty$

cost (lines 10-12), where the total cost is the sum of individual anonymization costs of all records within that cluster.

At this point, we introduce our clustering heuristic. We suggest that the *quality* of a cluster (implied by its cost) depends significantly on the choice of initial record T_a. If T_a happens to be an outlier (e.g., has far less or higher number of vertices than every other record in F, or its QIs are very rare) then even the best T_bs joining T_a's cluster will incur high costs of anonymization. Therefore on lines 13-22 of *ClusTree*, we perform the following check: We compute the mean and standard deviation of previously formed clusters (lines 14-15). If the cost of the newly formed cluster c is significantly higher than the mean, it is discarded and all records in c are inserted back to the input F. Otherwise, c can be added to the output R. We use χ to limit the discrepancy between the cost of c and the mean cost of clusters in R (line 16). χ is initialized to 0 (line 3) and incremented by the step size parameter s (line 25) at each iteration. We run iterations of the clustering procedure until χ goes above m (line 26), and afterwards we run one final pass with $\chi = +\infty$ (line 27) to allow clusters with any cost. The output of the clustering algorithm is a set of ℓ-diverse clusters. Records in F that are not placed in any cluster in R are fully suppressed (lines 6-8). *ClusTree* terminates when less than ℓ records remain in the input F (lines 6-8 and 23-24).

A lower value of χ sets a stricter upper bound on the costs of accepted clusters. The rate at which χ increases is determined by the input clustering parameter s. s should be small enough that expensive clusters are rejected in the first few iterations, but also large enough that clusters which were rejected in the previous iteration have a chance of being accepted in the next iteration. Also, smaller s implies larger number of passes over the input database F (although F is consumed in each iteration) and would hence be more time-consuming. The upper limit parameter m can be determined by experiment. However, if one assumes that clusters' costs will approximately follow a uniform distribution, the probability of a value falling outside $\mu+3*\sigma$ is significantly small (e.g., 99.7% of the samples in a normal distribution lie within 3 standard deviations of the mean). So, even in cases where costs are skewed or randomly distributed, a maximum upper limit of $m = 3$ or $m = 4$ should be reasonable.

5.4. Complexity Analysis

In this section we analyze the time complexity of our solution. We start with pairwise anonymization using *diversify* and *FindMapping*. Let our hierarchical data records have branching factor $b \geq 2$ and height h. For the sake of simplicity, we'll assume that all *mrca* operations, vertex generalizations and partial and full suppressions are performed in total time t per *diversify* call.

The greedy version of *FindMapping* requires $\frac{b\cdot(b+1)}{2}$ calls to *diversify* when called with two sets of subtrees. To anonymize all pairs of matched subtrees, *diversify* makes b recursive calls (lines 19-20 of Algorithm 1). Hence we obtain the following recurrence relation: $T(h) = \frac{b^2+b}{2} \cdot T(h-1) + b \cdot T(h-1) + t$ where $T(0) = t$. Solving this relation, we find that *diversify* with *FindMapping-GRD* is $\mathcal{O}(t \cdot \frac{b^{2h}}{2^h})$.

The LSAP version of *FindMapping* requires b^2 *diversify* calls to fill its cost matrix with agent-task costs (lines 3-7 in Algorithm 3), when called with two sets of subtrees. Then, finding an optimal solution to the LSAP using the Hungarian algorithm is $\mathcal{O}(b^3)$. Similar to above, *diversify* still makes b recursive calls to anonymize all pairs of matched subtrees. In this case we obtain the following recurrence relation: $T(h) = b^2 \cdot T(h-1) + t + \mathcal{O}(b^3) + b \cdot T(h-1)$ where $T(0) = t$. Solving this relation, we find that *diversify* with *FindMapping-LSAP* is $\mathcal{O}(b^{2h+1} + t \cdot b^{2h})$.

These results are significant in several ways. First, pairwise anonymization is exponential in h. Practical databases in real world, however, often have small h, e.g.,

$h = 3, 4$. Therefore this is not a pressing concern. Second, finding an optimal solution to a LSAP comes at the price of introducing an additional $\mathcal{O}(b^{2h+1})$ factor in asymptotic complexity. Third, there is the cost of performing generalizations and suppressions, which we denote by t. The efficiency of these operations is dependent on their implementation. Some operations can be implemented in constant time (e.g., checking if two vertices are 2-diverse, suppressing a given subtree). In our experiments we saw that the factor t has significant impact on execution time, hence efficient implementation of generalizations and suppressions is key to scalability.

The complexity analysis of our clustering algorithm *ClusTree* is as follows: Let n be the number of hierarchical data records in the database, ℓ be the ℓ-diversity parameter, and m and s be the clustering parameters in *ClusTree*. The complexity of pairwise anonymization depends on whether *GRD* or *LSAP* mapping is used, as shown above. We denote it here by $\mathcal{O}$*(diversify)*. We provide a worst-case analysis. The worst case occurs when the first cluster created is the least costly cluster possible, and therefore no cluster is accepted afterwards until the final iteration.

After initializing a cluster with a record, *ClusTree* (Algorithm 5) tries finding $\ell - 1$ other records to join that cluster (Algorithm 4). This requires going over the remaining records in the database $\ell - 1$ times and calling *diversify*. Hence *CreateCluster* is $\mathcal{O}(n \cdot \ell \cdot$ *diversify)*. Calculating a cluster's cost (lines 10-12) can be done cumulatively within *CreateCluster* while the cluster is being formed, and there are online algorithms to compute mean and variance [Welford 1962] so that computing and updating them when a new cluster is formed can be a constant time operation (lines 13-22). We therefore find that a single pass of *ClusTree* over its input database is $\mathcal{O}(n^2 \cdot \ell \cdot$ *diversify)*. A quick calculation shows that *ClusTree* performs $\lfloor \frac{m}{s} \rfloor + 2$ passes over the data, resulting in a time complexity of $\mathcal{O}((\lfloor \frac{m}{s} \rfloor + 2) \cdot n^2 \cdot \ell \cdot$ *diversify)*.

5.5. Proofs of Correctness

We now prove the correctness of the algorithm *ClusTree* (given in Algorithm 5), that is, we prove that the output of the algorithm is an ℓ-diverse anonymization of F. To do this, we first prove the correctness of the algorithm *diversify* (given in Algorithm 1) which acts as a building block in *ClusTree*.

Definition 5.1. We say a class representative T is ℓ-*diverse if each vertex in T contains exactly ℓ sensitive values.

COROLLARY 5.2. *If a class representative T is ℓ-*diverse then T is also ℓ-diverse.*

THEOREM 5.3. *Let T_1 and T_2 be ℓ_1 and ℓ_2-*diverse class representatives of equivalence classes D_1 and D_2 respectively. Then diversify on T_1 and T_2 creates a $(\ell_1 + \ell_2)$-*diverse representative for the anonymization of T_1 and T_2.*

PROOF. By induction:
Base Case: If the height of T_1 is 0, that is, T_1 is $\emptyset$, T_2 is suppressed. Since we will not have any vertex in T^*, T^* is a valid anonymization of both T_1 and T_2 and satisfies $(\ell_1 + \ell_2)$-*diversity.
Inductive Step: Let us denote a data record with height k as $T^{h=k}$. By inductive hypothesis, we assume *diversify* runs correctly for records with height at most $k-1$. That is, *diversify* on ℓ_1-*diverse $T_1^{h=i}$ and ℓ_2-*diverse $T_2^{h=j}$ creates an $(\ell_1 + \ell_2)$-*diverse representative for $i, j \in [0, k-1]$. We now prove the theorem for $T_1^{h=i}$ and $T_2^{h=j}$ where $i, j \leq k$.

We proceed with the proof as follows. We first show that the roots are properly diversified, that is, generalized to a $(\ell_1 + \ell_2)$-*diverse representative vertex (or suppressed

if diversification is not possible). We then show that the children of both trees are properly mapped, paired and diversified.

Diversification of the Root: Let a, b, and t be the roots of $T_1^{h=i}$, $T_2^{h=j}$, and T^* respectively. Due to the anonymization process enforced by *diversify*, if $t \neq \emptyset$, we have $t_{SA} = a_{SA} \cup b_{SA}$. If $a_{SA} \cap b_{SA} \neq \emptyset$, then t_{SA} does not satisfy $(\ell_1 + \ell_2)$-*diversity. In such a case, *diversify*, at lines 4-6, suppresses $T_1^{h=i}$ and $T_2^{h=j}$ and subsequently $T^* = \emptyset$. Suppressed T^* is an $(\ell_1 + \ell_2)$-*diverse anonymization. If $a_{SA} \cap b_{SA} = \emptyset$, at lines 7-10, QI values in both roots are generalized into the nearest common parent, thus QI-isomorphism of T^* is ensured at root level. Since $t_{SA} = a_{SA} \cup b_{SA}$, $|a_{SA}| = \ell_1$, and $|b_{SA}| = \ell_2$ then t_{SA} will contain $\ell_1+\ell_2$ sensitive values, thus satisfies $(\ell_1+\ell_2)$-*diversity.

Diversification of the Children: Let $C_A = \{A_1, \ldots, A_m\}$, $C_B = \{B_1, \ldots, B_n\}$, $C_T = \{T_1, \ldots, T_m\}$ are the subtrees attached to a, b, and t respectively and $m \leq n$. If C_A is empty, *diversify*, at lines 11-17, suppresses all trees in C_B, consequently $C_T = \emptyset$. In such a case, T^* will be composed of a single already-diversified root, thus satisfies $(\ell_1 + \ell_2)$-*diversity. If C_A is not $\emptyset$, either of the ***FindMapping*** functions are called. Both algorithms guarantee that every $A_i \in C_A$ is paired with some unique $B_j \in C_B$. *diversify*, at lines 19-20, calls itself recursively on the paired subtrees. Note that due to omission of the root, all subtrees in C_A and C_B have heights less than k. By the inductive hypothesis, for every pair (A_i, B_j) matched, *diversify* correctly returns $(\ell_1 + \ell_2)$-*diverse anonymization T_i of A_i and B_j. If there exists any unpaired subtree, *diversify*, in lines 21-23, suppresses it. Since the root is already-diversified, all vertices in T^* satisfies $(\ell_1 + \ell_2)$-*diversity. □

THEOREM 5.4. *Algorithm CreateCluster, when given records $F = \{T_1, \ldots, T_n\}$, $\ell \leq n$, $T_a \in F$ returns a cluster c where c_{rep} is ℓ-*diverse and $T_a \in c_{init}$.*

PROOF. At start, representative c_{rep} is initialized to T_a satisfying 1-diversity. At iteration i of the for loop, $(i-1)$-*diverse c_{rep} is diversified with a 1-diverse record and by Theorem 5.3, the resulting representative which is assigned to c_{rep} satisfies i-diversity. At the end of the loop, c_{rep} satisfies ℓ-*diversity (thus, ℓ-diversity) and c_{init} contains the associated equivalence class. □

THEOREM 5.5. *ClusTree, when called on records $F = \{T_1, \ldots, T_n\}$, returns an ℓ-diverse anonymization of F.*

PROOF. The *ClusTree* algorithm is basically a loop where at each iteration the following is performed:

— At lines 5-22, *ClusTree* scans all records currently in F once and for each record, the function *CreateCluster* creates a single ℓ-diverse cluster. Between lines 16-28, if the quality of the previously-formed cluster is far away from normal parameters, the cluster is discarded. Otherwise, it is moved from F to the result list. Distance threshold on the quality is controlled by the parameter χ.
— χ is incremented, and after reaching m it is set as ∞.

The algorithm halts only when there are less than ℓ records not clustered, in which case these records are suppressed. Due to correctness of *CreateCluster*, if the algorithm terminates, every record in F (except the few suppressed ones) belongs to exactly one cluster (equivalence class). Thus, by Definition 3.15, the returned clusters and the corresponding equivalence classes give an ℓ-diverse anonymization of the original records.

We conclude by stating that the algorithm always halts, that is, we will eventually have $|F| < \ell$. Note that the distance threshold χ that decides whether to discard a previously-formed cluster is monotonically increasing with each iteration of the while loop. After reaching m, χ is set to ∞. When this happens, no cluster will be discarded,

thus every cluster formed by *CreateCluster* function is removed from F. Since *CreateCluster* is called on every record in T_a, we will eventually be left with few enough records in F and the algorithm returns. □

6. EXPERIMENTAL EVALUATION

We implemented our algorithms in Java (v1.8.0) and used MongoDB (v2.4.7) to store our datasets. Experiments were conducted on a commodity machine with Intel i7 2.40 GHz CPU and 16 GB RAM.

Evaluation metrics. We use two means of evaluation: *LM cost* and *average query accuracy*. *LM* outputs a numerical value between 0 and 1 that conveys the average cost of generalizations and suppressions over the whole database. Lower *LM cost* implies higher data utility and therefore preferable anonymization.

For measuring *query accuracy*, we randomly generate several aggregate count queries (e.g., "How many students took *CS301*?" or "How many *CS* classes were taken in total?"). We issue these queries on the original dataset (X_i denotes the result of the ith query) and the anonymized dataset (Y_i denotes the result of the ith query). Then, average query accuracy is computed as follows (where N is the total number of queries):

$$\frac{\sum_{i=1}^{N}(1-(\frac{|Y_i-X_i|}{X_i}*100\%))}{N}$$

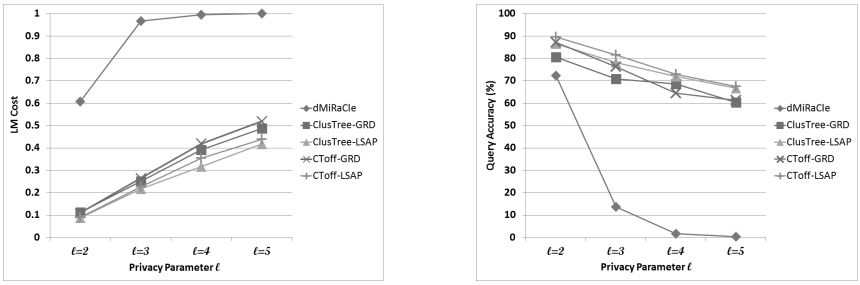
Datasets. We report results on three datasets (two *synthetic* and one *real*) obtained from two different universities in Turkey. Both datasets share a similar schema to that in Fig. 2.

For the *synthetic* datasets, we obtained data regarding students from Sabanci University's Computer Science (CS) program. The data contained the GPA and (partial) course grades of 30 students from this year's graduating class. To test with a meaningful number of data records, we simulated several students based on this sample, with the guidelines explained in the next paragraph.

We assumed that approximately the same number of students graduate every year, and set their current age according to their year of graduation. We simulated GPA values using a normal distribution, where the mean and the standard deviation were determined by the GPA scores of our sample. According to Sabanci University's CS program requirements, we ensured that all students took the obligatory classes. To each student, we randomly assigned a fixed number of classes from the pool of core classes, and a varying number of technical area electives. Students' grades were determined by their GPA and the type of class (e.g., we observed that most students perform better in obligatory classes). We assumed that a student would buy 0 to 2 books for each class.

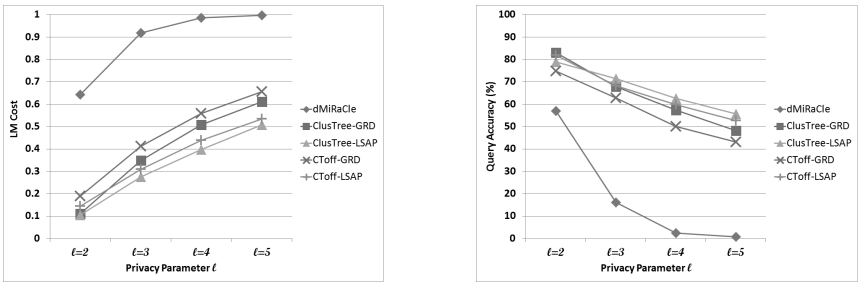
We created two synthetic datasets, *syntheticT* and *syntheticS*, both containing 1000 students with approximately 20 classes per student. *syntheticS* uses the schema in Fig. 2, i.e., (major,YoB) $\rightarrow$ classes $\rightarrow$ books. In order to test with an increased height, in *syntheticT* we added an intermediate level between the root and the classes, that depicts the year in which classes were taken, i.e., freshman, sophomore, junior or senior. Therefore the schema in *syntheticT* is: (major,YoB) $\rightarrow$ college years $\rightarrow$ classes $\rightarrow$ books. The division of classes into college years was probabilistic based on whether the class is a pre-requisite for any of the other classes the student took, and the usual timeframe in which the class is actually taken at Sabanci University.

The *real* dataset contains 3162 students together with their years of birth, their GPA, the classes they took and the grades they received. So, records in this dataset



LM cost Query accuracy

Fig. 10: Results on the *syntheticS* dataset for $\ell = 2, 3, 4, 5$



LM cost Query accuracy

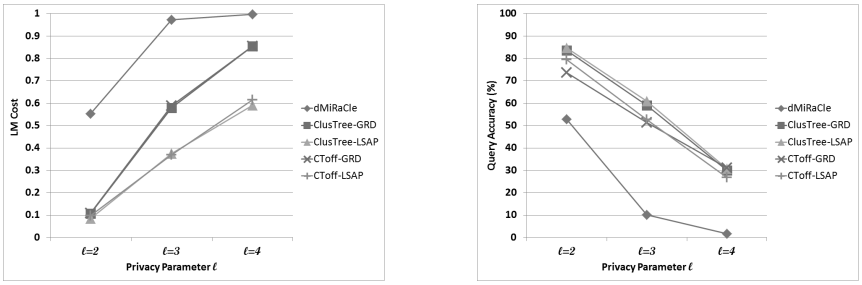
Fig. 11: Results on the *syntheticT* dataset for $\ell = 2, 3, 4, 5$

have only the first two levels shown in Fig. 2. Furthermore, instead of the QI attribute *age*, we used *year of birth*. We set DGHs of courses according to their IDs.

Algorithms. We evaluate five approaches, four of which are presented in this paper. For these, we used the LM metric as the anonymization cost metric in Algorithm 1. We tested *ClusTree* with the greedy and LSAP-based implementations of *FindMapping*. We call the resulting methods *ClusTree-GRD* and *ClusTree-LSAP*, respectively. Regarding the parameters of *ClusTree*, we set $s = 0.5$ since we saw that values below 0.5 did not make an observable difference, and we obtained the best results with $m = 4$.

To demonstrate the effectiveness of our clustering heuristic, we implemented a second procedure that does not contain any checks regarding the costs of clusters, i.e., we initialize $\chi = +\infty$ on line 3 of Algorithm 5. We refer to this implementation as *CToff*. There are also two versions of *CToff*: *CToff-GRD* and *CToff-LSAP*, depending on whether *FindMapping* is greedy or LSAP-based.

In addition, we implemented the multi-relational k-anonymity algorithm together with its ℓ-diversity extension (*dMiRaCle*) proposed in [Nergiz et al. 2009]. To the best of our knowledge, this is the only algorithm that can provide similar privacy guarantees to ours (i.e., ℓ-diversity) in hierarchical data. We converted our datasets into multi-relational databases and ran *dMiRaCle* on them. There are two parameters in *dMiRaCle*: *climit* and *th*. In our tests, we exactly mimicked the parameters suggested by the authors: We set *climit* to be 150 and tested with $th \in [0, 1]$ with increments of 0.1, and picked the best-performing result to report in this paper.



LM cost Query accuracy

Fig. 12: Results on the *real* dataset for $\ell = 2, 3, 4$

Results and Discussion. We graph our results for varying values of ℓ in Fig. 10, Fig. 11 and Fig. 12 on the synthetic datasets and the real dataset, respectively. In all experiments, we observe that the anonymization costs increase and query accuracy decreases as privacy requirements get stricter, i.e., ℓ is increased from 2 to 5. Two factors contribute to the loss of data utility when ℓ is increased: (1) The anonymization algorithm needs to find ℓ records for each cluster, i.e., higher ℓ requires more records per cluster. Each record that joins a cluster causes generalizations and/or suppressions. These anonymization operations are never reverted at a later point (e.g., when a new record joins a cluster), therefore the cost of a cluster always accumulates. (2) For large values of ℓ, it is harder to find ℓ different sensitive values per vertex. Consider a case where the instructor of *CS306* decided to grade very generously and all students received either *A* or *A-* from this class. When $\ell = 3$, *CS306* courses may never be matched with each other simply because there are only 2 different grades observed in the database. Hence, either all occurrences of *CS306* have to be suppressed, or they will be generalized with other classes (e.g., *CS3*** classes would be the best candidates) so that they become 3-diverse in the output.

We also observe that our algorithm outperforms *dMiRaCle* by a great margin in every experiment. As explained in Section 2, [Nergiz et al. 2009]'s *dMiRaCle* is primarily concerned with k-anonymity, and its ℓ-diversity extension depends on k-anonymizing an input dataset first and then finding an ℓ-diverse output. This can be a reasonable strategy when ℓ is small (e.g., $\ell = 2$), since a 2-anonymous equivalence class can, by coincidence, happen to be 2-diverse (or, making it 2-diverse might require very few operations). However, when $\ell = 3$ or 4, if the initial equivalence class is not built with ℓ-diversity in mind, later operations to make it ℓ-diverse will be very costly. Our experiments demonstrate this: There is a sharp increase in LM cost and a sharp decrease in query accuracy (in all three datasets) when ℓ is increased from 2 to 3.

We obtained better results on the synthetic datasets compared to the real dataset. We believe that this is caused by the fact that the real dataset is more sparse (e.g., there are 5000 unique classes, some of which are taken by very few students) and has more variance (e.g., some students took only 1-2 classes, whereas others took 60-70). In contrast, the synthetic datasets are more evenly distributed, e.g., all students are CS majors that take around the same number of classes, most of which are classes in Computer Science or related areas. Also, we obtain roughly 10-15% better results on *syntheticS* compared to *syntheticT*. The probable cause for this is the division of classes into college years in *syntheticT*. For example, consider two students $S1$ and $S2$ that take the elective class *EL101*, but $S1$ takes *EL101* in her freshman year whereas $S2$

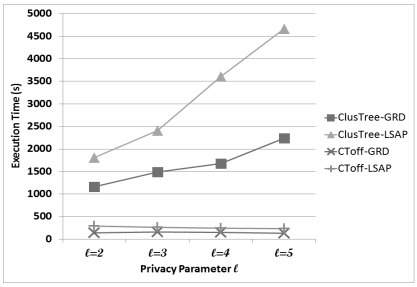


Fig. 13: Execution time on the *syntheticS* dataset

takes *EL101* in her senior year. Unless *diversify* decides to match $S1$'s freshman year with $S2$'s senior year (which is a small probability, assuming $S1$'s freshman classes are more similar to $S2$'s freshman classes rather than her senior classes) the *EL101* vertices will not be matched with each other. Instead, they will be matched with other classes or suppressed, due to the top-down nature of *diversify*. The *syntheticS* dataset does not suffer from this problem, since classes are directly children of the root vertices, and are not divided into college years.

We also would like to study the effects of our heuristics, by comparing (1) *ClusTree* versus *CToff* to validate that our clustering heuristic is useful, and (2) *LSAP* versus *GRD* to validate the effect of using an optimal solution against a greedy solution. In most experiments, we see that *ClusTree* outperforms *CToff* and *LSAP* outperforms *GRD*, as expected. The difference between *LSAP* and *GRD* is usually more evident when ℓ is large, apart from the $\ell = 4$ case on the *real* dataset, since most of the data in this experiment is destroyed no matter which algorithm is used. Also, although our *LSAP* approach provides an optimal solution to the subtree matching problem, neither *ClusTree* nor *CToff* guarantee optimality in the clustering phase - as in any clustering algorithm. Therefore we cannot claim that *ClusTree-LSAP* or *CToff-LSAP* are optimal or they should outperform their greedy counterparts in all experiments. In most experiments they do, which is intuitive, but there are also cases where the *GRD* approach performs almost as good as or somewhat better than *LSAP* (this happens often when ℓ is small).

With regard to efficiency, we obtained the execution times in Fig. 13 on *syntheticS*. This dataset contains 1000 hierarchical data records (with height = 3) and a total of approximately 42000 vertices. *CToff* is significantly faster than *ClusTree*, since it performs a single pass over the data. For all ℓ values, it took 2-3 minutes to run *CToff-GRD* and 3-4 minutes to run *CToff-LSAP*. Since *ClusTree* performs multiple passes over the data (with increasing χ), it turns out to be roughly 10-15 times slower than *CToff*. The differences between execution times become more significant as ℓ is increased. All of these results are in line with our complexity analyses.

Finally, we would like to emphasize the trade-offs between data utility and efficiency. The choice of using *ClusTree* over *CToff* and *LSAP* over *GRD* both increase data utility, but come at the cost of increased execution time. On average, the best-performing algorithm (in terms of query accuracy and LM cost) is *ClusTree-LSAP*, which also happens to be the slowest.

7. CONCLUSION

In this paper we investigated the problem of privacy in hierarchical data publishing. We discussed how popular privacy notions such as k-anonymity and ℓ-diversity can be applied to hierarchical data. We designed an algorithm that produces ℓ-diverse anonymizations of collections of hierarchical data. Our algorithm is independent of the domains and generalization hierarchies of attributes, and the anonymization cost metric used. Even though we use the LM metric in this paper, our approach is suitable for other monotonic cost metrics. For example, one can use a metric that penalizes certain levels in the hierarchical schema more than others (e.g., to apply more emphasis on courses than evaluations). Other domain-specific heuristics can also be employed. To fight sparsity of high-dimensional data and provide flexibility, our solution uses local recoding. We also address negative knowledge as well as positive knowledge: For every piece of information an adversary has (e.g., student has taken class X and/or has not taken class Y), there are at least ℓ records in the anonymized output that fit this description. Therefore, the adversary's confidence regarding a particular sensitive value of his victim is always bounded by $1/\ell$.

There are also certain limitations of our approach. For example, if all records in an equivalence class contain the letter grade A for different classes, an adversary may learn with probability $> 1/\ell$ that his victim has received an A from *some* class, even though he cannot be certain *which* class it was. In certain cases such disclosures might be unacceptable, e.g., adversary learns that his victim has AIDS from an anonymized medical dataset. To aid this problem, one can easily extend our algorithm to prohibit certain sensitive values from appearing multiple times in a class representative.

There exist several directions for future work. First, our anonymization strategy does not allow adding noise to the output. One could try to see whether data utility can be improved by adding noise and counterfeits. Second, different definitions of privacy (e.g., differential privacy [Dwork 2008], that relies on noise addition) can be applied to hierarchical data. Third, there are numerous tools and engines that process hierarchical data. In particular, XML streams and query engines are widely used in today's world. An interesting area of research is how our definitions of privacy can be applied in these contexts (e.g., XML data streams) [Zhou et al. 2009].

REFERENCES

Roberto J Bayardo and Rakesh Agrawal. 2005. Data privacy through optimal k-anonymization. In *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*. IEEE, 217–228.

Elisa Bertino, Silvana Castano, Elena Ferrari, and Marco Mesiti. 2000. Specifying and enforcing access control policies for XML document sources. *World Wide Web* 3, 3 (2000), 139–151.

Elisa Bertino, Dan Lin, and Wei Jiang. 2008. A survey of quantification of privacy preserving data mining algorithms. In *Privacy-preserving data mining*. Springer, 183–205.

James Cheng, Ada Wai-Chee Fu, and Jia Liu. 2010. K-isomorphism: privacy preserving network publication against structural attacks. In *Proceedings of the 29th ACM International Conference on Management of Data (SIGMOD 2010)*. ACM, 459–470.

A Ercument Cicek, Mehmet Ercan Nergiz, and Yucel Saygin. 2014. Ensuring location diversity in privacy-preserving spatio-temporal data publishing. *The VLDB Journal* 23, 4 (2014), 609–625.

Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. 2002. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security (TISSEC)* 5, 2 (2002), 169–202.

Cynthia Dwork. 2008. Differential privacy: A survey of results. In *Theory and applications of models of computation*. Springer, 1–19.

Irini Fundulaki and Maarten Marx. 2004. Specifying access control policies for XML documents with XPath. In *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies*. ACM, 61–69.

Benjamin Fung, Ke Wang, Rui Chen, and Philip S Yu. 2010. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (CSUR)* 42, 4 (2010), 14.

Gabriel Ghinita, Panos Kalnis, and Yufei Tao. 2011. Anonymous publication of sensitive transactional data. *IEEE Transactions on Knowledge and Data Engineering* 23, 2 (2011), 161–174.

Olga Gkountouna and Manolis Terrovitis. 2015. Anonymizing Collections of Tree-Structured Data. (2015).

Jiawei Han, Micheline Kamber, and Jian Pei. 2011. *Data mining: concepts and techniques*. Elsevier.

Yeye He and Jeffrey F Naughton. 2009. Anonymization of set-valued data via top-down, local generalization. *Proceedings of the VLDB Endowment* 2, 1 (2009), 934–945.

Vijay S Iyengar. 2002. Transforming data to satisfy privacy constraints. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD 2002)*. ACM, 279–288.

Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 2, 1-2 (1955), 83–97.

Anders H Landberg, Kinh Nguyen, Eric Pardede, and J Wenny Rahayu. 2014. δ-Dependency for privacy-preserving XML data publishing. *Journal of Biomedical Informatics* 50 (2014), 77–94.

Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. 2005. Incognito: Efficient full-domain k-anonymity. In *Proceedings of the 24th ACM International Conference on Management of Data (SIGMOD 2005)*. ACM, 49–60.

Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. 2006. Mondrian multidimensional k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE 2006)*. IEEE, 25–25.

Junqiang Liu and Ke Wang. 2010. On optimal anonymization for l+-diversity. In *Proceedings of the 26th International Conference on Data Engineering (ICDE 2010)*. IEEE, 213–224.

Kun Liu and Evimaria Terzi. 2008. Towards identity anonymization on graphs. In *Proceedings of the 27th ACM International Conference on Management of Data (SIGMOD 2008)*. ACM, 93–106.

Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. 2007. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 3.

Adam Meyerson and Ryan Williams. 2004. On the complexity of optimal k-anonymity. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems (PODS 2004)*. ACM, 223–228.

James Munkres. 1957. Algorithms for the assignment and transportation problems. *J. Soc. Indust. Appl. Math.* 5, 1 (1957), 32–38.

Mehmet Ercan Nergiz, Maurizio Atzori, and Yucel Saygin. 2008. Towards trajectory anonymization: a generalization-based approach. In *Proceedings of the 2008 ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*. ACM, 52–61.

Mehmet Ercan Nergiz, Christopher Clifton, and Ahmet Erhan Nergiz. 2009. Multirelational k-anonymity. *IEEE Transactions on Knowledge and Data Engineering* 21, 8 (2009), 1104–1117.

Mehmet Ercan Nergiz, Acar Tamersoy, and Yucel Saygin. 2011. Instant anonymization. *ACM Transactions on Database Systems* 36, 1 (2011), 2.

David W Pentico. 2007. Assignment problems: A golden anniversary survey. *European Journal of Operational Research* 176, 2 (2007), 774–793.

Pierangela Samarati and Latanya Sweeney. 1998. *Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression*. Technical Report. SRI International.

Latanya Sweeney. 2000. *Uniqueness of simple demographics in the US population*. Technical Report. Carnegie Mellon University.

Latanya Sweeney. 2002a. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 05 (2002), 571–588.

Latanya Sweeney. 2002b. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 05 (2002), 557–570.

Acar Tamersoy, Grigorios Loukides, Mehmet Ercan Nergiz, Yucel Saygin, and Bradley Malin. 2012. Anonymization of longitudinal electronic medical records. *IEEE Transactions on Information Technology in Biomedicine* 16, 3 (2012), 413–423.

Manolis Terrovitis and Nikos Mamoulis. 2008. Privacy preservation in the publication of trajectories. In *Proceedings of the 9th International Conference on Mobile Data Management (MDM 2008)*. IEEE, 65–72.

Manolis Terrovitis, Nikos Mamoulis, and Panos Kalnis. 2008. Privacy-preserving anonymization of set-valued data. *Proceedings of the VLDB Endowment* 1, 1 (2008), 115–125.

Manolis Terrovitis, Nikos Mamoulis, and Panos Kalnis. 2011. Local and global recoding methods for anonymizing set-valued data. *The VLDB Journal* 20, 1 (2011), 83–106.

Traian Marius Truta and Bindu Vinay. 2006. Privacy protection: p-sensitive k-anonymity property. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE 2006) Workshops*. IEEE, 94.

BP Welford. 1962. Note on a method for calculating corrected sums of squares and products. *Technometrics* 4, 3 (1962), 419–420.

Xiaokui Xiao and Yufei Tao. 2006a. Anatomy: Simple and effective privacy preservation. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB 2006)*. VLDB Endowment, 139–150.

Xiaokui Xiao and Yufei Tao. 2006b. Personalized privacy preservation. In *Proceedings of the 25th ACM International Conference on Management of Data (SIGMOD 2006)*. ACM, 229–240.

Xiaokui Xiao, Ke Yi, and Yufei Tao. 2010. The hardness and approximation algorithms for l-diversity. In *Proceedings of the 13th International Conference on Extending Database Technology (EDBT 2010)*. ACM, 135–146.

Xiaochun Yang and Chen Li. 2004. Secure XML publishing without information leakage in the presence of data inference. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB 2004)*. VLDB Endowment, 96–107.

Elena Zheleva and Lise Getoor. 2008. Preserving the privacy of sensitive relationships in graph data. In *Proceedings of the 2nd International Workshop on Privacy, Security and Trust in KDD (PinKDD 2008)*. Springer, 153–171.

Bin Zhou, Yi Han, Jian Pei, Bin Jiang, Yufei Tao, and Yan Jia. 2009. Continuous privacy preserving publishing of data streams. In *Proceedings of the 12th International Conference on Extending Database Technology (EDBT 2009)*. ACM, 648–659.

Bin Zhou, Jian Pei, and WoShun Luk. 2008. A brief survey on anonymization techniques for privacy preserving publishing of social network data. *ACM SIGKDD Explorations Newsletter* 10, 2 (2008), 12–22.