University of California
Los Angeles

# Preserving Privacy in Hierarchical Data Publishing

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Computer Science

by

**Mehmet Emre Gursoy**

2015

Abstract of the Thesis

# Preserving Privacy in Hierarchical Data Publishing

by

**Mehmet Emre Gursoy**

Master of Science in Computer Science

University of California, Los Angeles, 2015

Professor Peter L. Reiher, Co-chair

Professor Carlo Zaniolo, Co-chair

Many applications today rely on storage and management of semi-structured information. Data stored in such databases often has to be shared with untrusted third parties for research-related or other purposes, which makes privacy a fundamental problem. In this thesis, we study privacy-preserving publishing of hierarchical data, where an individual's information in a database can be collected and represented using a tree structure. We provide practical algorithms and solutions regarding how some of the well-known privacy definitions (e.g., $k$-anonymity and $l$-diversity), which were designed for tabular data, can be generalized and applied to hierarchical data. We show that different solutions are preferable depending on the needs and expectations of data recipients, in order to maximize data applicability and utility. Thus, we introduce a range of techniques to anonymize hierarchical data. We experiment on realistic synthetic databases to test our algorithms and validate our claims.

The thesis of Mehmet Emre Gursoy is approved.

Junghoo Cho

Wei Wang

Carlo Zaniolo, Committee Co-chair

Peter L. Reiher, Committee Co-chair

University of California, Los Angeles

2015

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgments

First and foremost, I would like to thank my advisor, Professor Peter Reiher, for his support and guidance throughout my time at UCLA. It was a privilege for me to get my first research experience under his excellent guidance, and he has inspired me greatly. A big thank you to my committee members who have been very welcoming and open in supervising this thesis.

I would also like to thank my family, especially my mom and dad, for their unconditional love and continuous support of my education. I could not have gotten this far without them. I am grateful to my friends that are scattered all around the world for keeping in touch and entertaining me whenever I needed a break from work. A special thanks to my friends in LA for all the fun we had together. I was also fortunate to have a number of friends in various cities in the United States, which gave me the chance to travel to and see some of the most amazing places in the world.

I would like to express my gratitude to the members of Laboratory for Advanced Systems Research (LASR), and all others whom I had the chance of collaborating with in the last couple of years. I am indebted to Professor Yucel Saygin and Professor Ercan Nergiz for introducing me to the topic of privacy-preserving data publishing and offering advice on a significant portion of this work.

# CHAPTER 1

# Introduction

Widespread use of computers and networked devices has contributed to various things, including the ease of collecting and storing person-specific microdata. Entities such as governments, industrial and educational organizations may need to share their data with untrusted third parties for research related or other purposes. Oftentimes the data contains sensitive information that can violate individuals' privacy, and therefore precautions need to be taken before the data can be published to a third party. Researchers working in *privacy-preserving data publishing* and *data anonymization* have been trying to find answers to questions such as how these precautions can be defined, how data can be modified to meet these definitions and goals, and whether data utility can be conserved while an "appropriate" level of privacy is satisfied [17].

A typical scenario is presented in Fig. 1.1. Alice, Bob, Charlie and Dave are some patients that had to visit the hospital due to certain health problems. The hospital has a database that keeps track of the diagnoses its patients receive. Upon being treated, data from these four individuals have been stored in the hospital's database. Our assumption at this point is that the data owner and publisher (in this case, the hospital) are trustworthy, i.e., there are no "insider threats". Later, the hospital decides to share its data with a university, so that researchers can use this data to test their hypotheses. Can this third party be trusted? Ideally, the hospital needs to make sure that the data it provides is properly anonymized so that an attacker cannot link individuals to their records in the database.

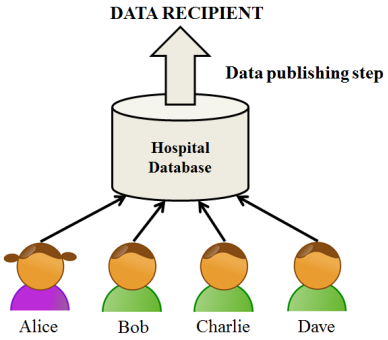



Figure 1.1: Collecting and sharing individuals' sensitive data

One popular way of dealing with this problem is data perturbation, which distorts the data by adding random noise to existing records [6], swapping values of fields [40] and/or supplying fake records based on statistical patterns observed in the data. However, we will focus on cases where the data publisher would like to be truthful, such that all records in the published dataset correspond to a real life entity/occurrence. This motivates the need for proper *anonymization* [12].

The least one can do to protect privacy is to remove explicitly identifying information (e.g., social security number, name) from a published dataset. This is obviously a necessary step, but history has shown that it is not sufficient: An early study [42] reports that gender, date of birth and 5-digit zip code are enough to uniquely identify 87% of the United States population. In another work, it has been shown that the triplet (gender, date of birth, zip code) can be used to link voter registration records and medical data published by an insurance company. The result reveals the medical records of the governor of Massachusetts [43]. Somewhat recently, researchers from UT Austin showed that a so-called anonymized Netflix dataset, made available for research, could be used to reveal political preferences and other sensitive information about Netflix users [33].

An adversarial attempt is often carried out using background information and quasi-identifiers (abbreviated "QI") of individuals, i.e., attributes that do not necessarily disclose individuals' identity when used alone, but can be used in

| Name | Age | Gender | Zip Code |
|---|---|---|---|
| Alice | 24 | female | 92620 |
| Jason | 28 | male | 90024 |
| James | 32 | male | 96000 |
| Mary | 35 | female | 90105 |

(a) Public dataset

| Age | Gender | Zip Code | Condition |
|---|---|---|---|
| 24 | female | 92620 | HIV |
| 28 | female | 92440 | HIV |
| 30 | male | 92451 | flu |
| 36 | male | 92655 | asthma |

(b) Hospital database w/ sensitive info

| Age | Gender | Zip Code | Condition |
|---|---|---|---|
| 2* | female | 92*** | HIV |
| 2* | female | 92*** | HIV |
| 3* | male | 92*** | flu |
| 3* | male | 92*** | asthma |

(c) 2-anonymous hospital database

| Age | Gender | Zip Code | Condition |
|---|---|---|---|
| $\leq 50$ | * | 926** | HIV |
| $\leq 50$ | * | 924** | HIV |
| $\leq 50$ | * | 924** | flu |
| $\leq 50$ | * | 926** | asthma |

(d) 2-diverse hospital database

Figure 1.2: Record linkage attacks and possible defenses

combination and/or together with external databases to single out record owners. Gender, nationality, date of birth and zip code are popular examples. Usage of quasi-identifiers to identify records and infer sensitive values is often referred to as a linkage attack [17].

Let us demonstrate the feasibility of privacy attacks through QIs using a fictitious example. Assume that Alice's information can be found in a public database, such as the one in Fig. 1.2a, or an adversary knows Alice in real life and therefore has information regarding her age, gender and the location of her home. Alice, Bob, Charlie and Dave all visited the same hospital, and the hospital logged their health conditions. Before publishing this sensitive database, the hospital thought that it would be good idea to anonymize it first, hence removing everyone's explicit identifiers (e.g., names). They ended up with the table in Fig. 1.2b. The adversary knows where Alice was hospitalized, so he starts searching for her record. Since only the first tuple in 1.2b can refer to Alice, the adversary learns that Alice has HIV.

One of the most widely studied and well-known privacy protection methods is $k$-anonymity [43][41]. In a $k$-anonymous table, for each existing combination of QI values, at least $k$ records in the dataset share the same combination [13]. In other words, each record is QI-wise indistinguishable from at least $k-1$ other records. This anonymization occurs via generalizations (i.e., replacing specific values by more general ones) and suppressions (i.e., deleting some values or tuples) [11]. Fig. 1.2c shows a 2-anonymization of the sensitive dataset. In this case, an attacker can find that either the first tuple or the second belongs to Alice, but has no way of telling exactly which one.

By definition, $k$-anonymity does not take into account the distribution of sensitive values. For instance, upon linking Alice to the first two tuples in Fig. 1.2c, the attacker may not learn which record corresponds to Alice; but he may still learn that Alice suffers from HIV since both records have the same sensitive information. This is a clear violation of Alice's privacy, and is definitely undesirable.

Several sources acknowledge this problem, and point to possible countermeasures such as bounding an attacker's confidence of inferring a sensitive attribute using a publisher-specified threshold [45][47][27]. A popular and effective solution is $l$-diversity [31]. In its simplest form, $l$-diversity ensures that every QI-wise indistinguishable block contains $l$ sensitive values that are "distinct" from one another, so that a sensitive attribute is "well-represented". (There can be other definitions for being "well-represented", e.g., probabilistic definitions.) Notice that $l$-anonymity is a prerequisite for $l$-diversity. Fig. 1.2d is 2-diverse, and therefore unlike the 2-anonymous database, the adversary can no longer determine whether Alice has HIV or asthma.

Most of the notions of privacy have been invented for and applied to tabular (i.e., relational) data. However, the era of big data is certainly increasing the emphasis on collection of semi-structured data. This thesis focuses on hierarchically structured data, that is, cases where an individual's presence in a database can

be abstracted as a tree. XML documents, some spatio-temporal databases and data warehouses can be represented hierarchically [37]. For example, a patient may visit the same hospital several times receiving different diagnoses each time and receiving applicable medication for each diagnosis. In this case, the patient's demographic information can be kept at the root, diagnoses can be placed as children of the root and similarly, each medication can be associated with the corresponding diagnosis.

Data formats suitable for representing hierarchical structures and trees are manifold: XML, JSON and YAML are popular examples. Emergence of document-oriented databases following the NoSQL trend (e.g., MongoDB) has contributed to storage and management of structured data. Relational databases with snowflake schemas can also store hierarchical data. In this paper we consider hierarchical data, regardless of how it is stored. Our representation and discussion applies to all storage schemes outlined above.

The objective of our study is to properly define existing notions of privacy on hierarchical data. We then discuss practical methods that enforce these privacy constraints. Anonymization causes loss in data utility, which we measure by extending popular metrics from the literature. Our algorithms aim to maximize data utility while performing anonymization, but they are not dependent on the choice of cost metric. Finally, we take a look at privacy-preserving partial publishing of hierarchically structured multi-relational databases.

## 1.1 Related Work

Optimal $k$-anonymity has been shown to be NP-hard [32][26]. Due its popularity, though, significant research has been put into implementing efficient algorithms to achieve $k$-anonymity, most of which are based on heuristic approaches (e.g., approximation algorithms) and are suitable for tabular data, e.g., [5][22][25][44][2][38].

$l$-diversity requires $l$-anonymity, therefore the problem of computing $l$-diverse tables with minimum information loss is also computationally challenging. [50] shows that achieving optimal $l$-diversity through generalizations is NP-hard for $l \geq 3$. Among notable $l$-diversity algorithms are the ones presented in [31], [50] and [29]. There are also other definitions of privacy, some of which are stricter than the ones we will use, and others that have different requirements: $\delta$-presence [35], $t$-closeness [28] and *anatomy* [49] are some of the prevalent ones.

Even though multi-relational data mining [14] and extensions of single-tabular methods to support multi-relational databases is an active area [20][16], hierarchical data privacy has not been adequately studied. [51] demonstrates that semi-supervised learning algorithms may lead to privacy leakage in multi-relational databases. For privacy preservation, one of the earlier studies considers multi-relational $k$-anonymity [37]. This is closely connected to our work, and we provide a generalized version of the setting presented in this paper. Some of our anonymity algorithms are applicable to their setting, and we will make comparisons. There are other sources (e.g., [53]) that allow several tuples in a database to originate from the same individual, yet their discussion is limited to a single relation.

Related to our work are also [23] and [24]. [23] proposes to use a variation of $k$-anonymity to control users' access to XML databases. In contrast, we assume that an adversary can fully access a database once it is published. In [24], authors use *anatomy* [49] to anonymize medical data and electronic health records that are represented in XML format. Anatomy releases quasi-identifiers and sensitive values in seperate tables to eliminate linkage. On the other hand, our work is explicit regarding which sensitive value corresponds to which data record. Furthermore, our solutions are applicable to various scenarios of publishing and ways of storage.

Relevant literature discusses the applicability of privacy notions such as $k$-anonymity to a number of settings, including transactional databases and set-

valued data [21][48], spatio-temporal databases and trajectory data [34][10], graph data (where not only nodes but also edges, i.e., links between data, are required to be anonymized) [52][30] and social networks [54]. The success of these recent attempts motivated us to take a similar approach for hierarchical data anonymization.

## 1.2 Contributions

In this thesis, we make the following contributions:

- We formalize the notions of $k$-anonymity and $l$-diversity on hierarchical data. We explain why single-table methods are insufficient to preserve individuals' privacy in hierarchical databases.
- We provide a clustering-based solution to achieve $k$-anonymity on hierarchical data. This solution does not depend on any assumptions regarding the underlying dataset, or the utility cost metric used to compute information loss. Our algorithm is shown to outperform the state of the art multi-relational $k$-anonymity algorithm in [37].
- We enhance our $k$-anonymity algorithm to support $l$-diversity. We propose extensions, which our evaluations show to be effective. We then compare the data utility of $k$-anonymized and $l$-diversified datasets, in order to analyze if the additional privacy $l$-diversity provides is worth the increased utility cost.
- During all of the above, we assume explicit connections between records belonging to an individual. We then relax this assumption to support cases where "data ownership" is irrelevant for the data recipient (i.e., the end receiver of published data). We formalize this problem on multi-relational databases and illustrate its usefulness.

- We provide a framework that anonymizes such datasets. We show that this solution can be much more efficient and accurate for the relaxed problem in hand compared to the previous $k$-anonymity solution.

## 1.3 Organization

The remainder of this thesis is organized as follows: An introduction to hierarchical data is given in the next chapter. Our introduction is based on multi-relational databases with snowflake schemas rather than XML/JSON object representations, not only because XML/JSON is trivial to understand and visualize, but also because our definitions in Chapter 2 are re-visited in Chapter 5. We solve the problem of $k$-anonymity on hierarchical data in Chapter 3, providing detailed explanations of our functions and algorithms. We elaborate on our design choices and evaluate our algorithms. In Chapter 4, we point out the problems with "anonymity" and define the requirements for "diversity". We incorporate the notion of diversity into our solution, which appears to be significantly costly. However, by extending our clustering algorithm, we can decrease these costs by 30-40%. In Chapter 5, we give a brief recap of our assumptions in previous chapters, and explain how/when they can be relaxed. We undertake a problem that we call "partial publication": publishing only certain portions/relations of a multi-relational database. We show how this problem is different than the previous ones, and why less costly solutions could be found. We implement a solution that produces end results that preserve privacy and have low information loss. The final chapter concludes this thesis and discusses potential directions for future work.

# CHAPTER 2

# Problem Definition

## 2.1 Introduction to Hierarchical Data

We first provide formal definitions for the concepts considered in the previous section and the basics of what will be covered in the remainder of this thesis. We establish our initial discussion using multi-relational databases.

**Definition 1** ***(Relation)*** *A relation is a collection of tuples, in which each tuple follows a certain schema* $S = (A_1, A_2, ..., A_n)$*, where attributes* $A_1, .., A_n$ *are categorized under one of:*
*- Explicit Identifiers (e.g., name, SSN). In that case they are projected out before publication (and therefore from this point onwards we will ignore them).*
*- Quasi Identifiers (QI), which will be subject to generalizations and suppressions.*
*- Sensitive Attributes (SA), which will be left untouched.*
*- Join Keys (primary and foreign keys), which may be suppressed.*
*- Irrelevant attributes that bear no privacy risks, which can either stay or be projected out (from this point onwards, these will be ignored too).*

Join keys are used to explicitly connect records to their ancestors and children, and preserve the hierarchical structure of data. If join keys are generalized, we may lose whom a tuple belongs to (see Fig. 2.3 and Fig. 2.4 for examples). Therefore, only suppressions of join keys are allowed, and that may only happen when a record (and its subtree, if applicable) is being fully deleted (i.e., suppressed).

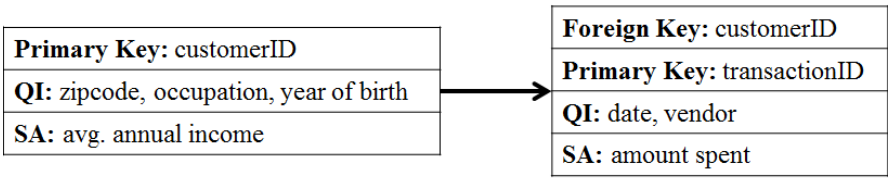



Figure 2.1: Schema for transaction data.

As discussed earlier, we limit our study to multi-relational schemas that satisfy certain constraints, from hereon to be referred to as **snowflake schema**s. We now explain relevant properties of our setting.

**Definition 2** ***(Central relation)*** *A relation $R$ is called the central relation of a snowflake schema, if and only if $R$ contains a primary key (a single attribute, e.g., customerID; or a collection of attributes that qualify to be the primary key), such that the primary key uniquely identifies (i.e., refers to) exactly one individual in the population.*

A snowflake schema should contain only one central relation. All relations should have one foreign key, whereas the central relation has no foreign keys. In our previous definition, we considered join key and QI attributes to be exclusive of one another. In cases where they are not (i.e., a join key is also a QI), one may trivially add a new attribute that copies the join key, and treat the new attribute as a QI. Figures 2.1 and 2.2 contain examples of snowflake schemas.

**Definition 3** ***($k$-anonymity across multiple relations)*** *Let $MR$ be multi-relational database with a snowflake schema. $MR$ has a central relation $CR$ with primary key $PK$, and multiple other relations: $MR = (CR, R_1, R_2, ..., R_n)$. Let $MR^*$ contain a transformed version of the data in $MR$. We say that $MR^*$ is a $k$-anonymized version of $MR$, if:*

*- $MR^*$ fully preserves the schematical structure of $MR$. That is, for every relation in $MR$, there is a relation in $MR^*$ that has the same join key, quasi-identifier*

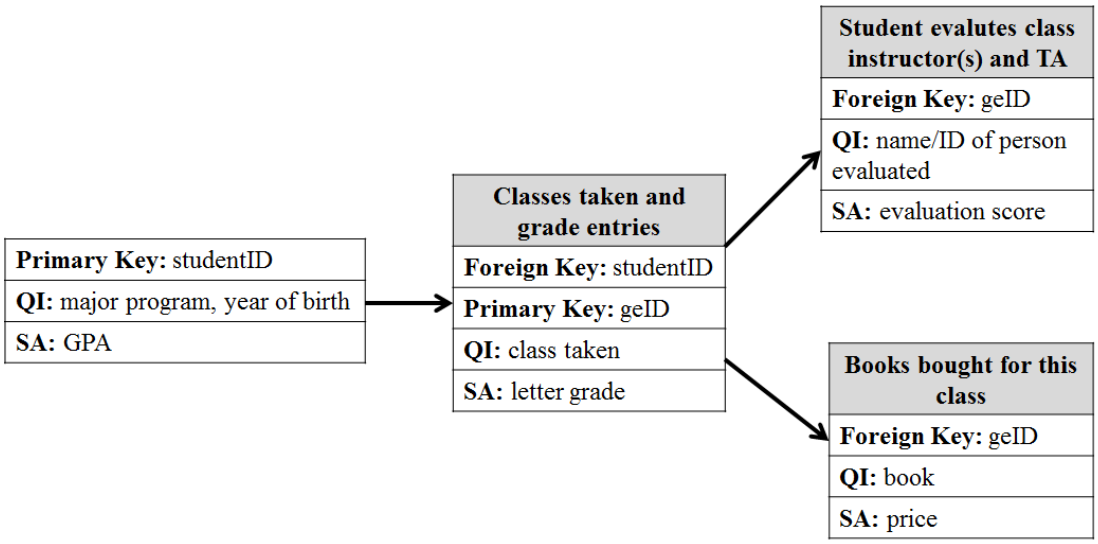



Figure 2.2: Schema for education data.

*and sensitive attributes. We denote the result* $MR^* = (CR^*, R_1^*, R_2^*, ..., R_n^*)$.
*- Let* $U^*$ *denote the universal join of all relations in* $MR^*$*, computed using equi-joins on join keys (primary and foreign keys), i.e.:* $U^* = CR^* \bowtie R_1^* \bowtie R_2^* ... \bowtie R_n^*$*. Let* $\sigma_{QI}$ *denote a legitimate selection query on* $U^*$*, constructed using quasi-identifier attributes in one or more relations. Then, for all possible* $\sigma_{QI}$*, the query* $\pi_{PK}(\sigma_{QI}(U^*))$ *should return either zero tuples, or at least* $k$ *distinct tuples.*

We allow no counterfeits (i.e., noise cannot be introduced to the data), therefore the following requirement should be added to the definition above:
*-* $MR^*$ *is "correct": For all relations* $R^* \in MR^*$*, each tuple* $t_i^* \in R^*$ *is a properly generalized or suppressed version of a tuple* $t_i \in R$*, where* $R \in MR$.

## 2.2 Notation

Similar to how a snowflake schema can be represented (or how an XML object can be viewed), we use a tree structure to visualize the data. For example, Fig. 2.3 contains data from three of the tables in Fig. 2.2. Converting this data into our

| studentID | major | YoB | GPA |
|---|---|---|---|
| S1 | Computer Science | 1994 | 3.26 |
| S2 | Computer Science | 1995 | 3.48 |

| studentID | geID | class | grade |
|---|---|---|---|
| S1 | SG1 | CS111 | B |
| S1 | SG2 | CS131 | A |
| S1 | SG3 | CS143 | A- |
| S2 | SG4 | CS111 | A- |
| S2 | SG5 | CS132 | A- |

| geID | Instructor / TA name | Evaluation (/10) |
|---|---|---|
| SG1 | Eggert | 7/10 |
| SG1 | TA1 | 6/10 |
| SG2 | Millstein | 9/10 |
| SG2 | TA7 | 9/10 |
| SG3 | TA4 | 8/10 |
| SG4 | Eggert | 6/10 |
| SG4 | TA2 | 9/10 |
| SG5 | Palsberg | 10/10 |

Figure 2.3: Sample education data.

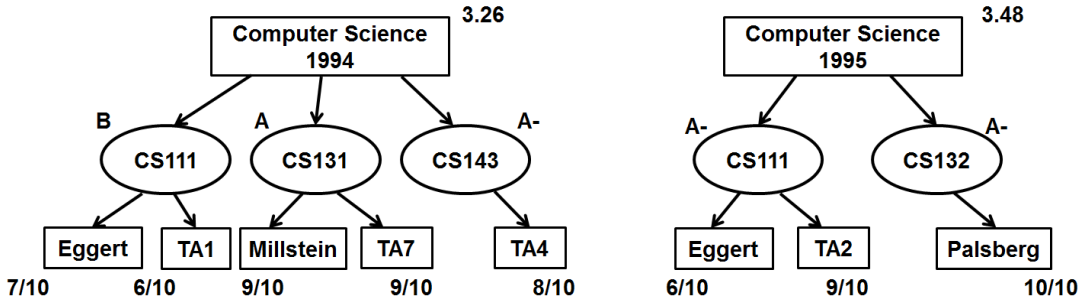



Figure 2.4: Converting data in multiple relations into trees.

notation, we get the trees in Fig. 2.4. Each tuple is represented using one tree node where join keys are used to establish connections between affiliated nodes. Quasi-identifiers are written as labels within nodes, and sensitive values sit right beside the nodes they are associated with. The shape of a tree node in our example has no significance (they are just there for extended clarity regarding levels and depth). We feel that it is trivial to encode the same information in XML (or any other markup language that can be used to store hierarchical data, e.g., JSON), hence we do not provide examples for these cases.

Next, we should define the notion of $k$-anonymity on tree-structured data. One can see that Definition 3 is essentially an application of this, targeting multi-relational databases. The definition that we give now is concerned with all types of hierarchical data. It should be noted that the trees we have are rooted and labeled. Root nodes always originate from the central relation, which means there

should be one tree per individual in our database. The ordering of siblings bears no significance (although for clarity we mimicked the order in Fig. 2.3 while drawing Fig. 2.4), i.e., trees are unordered.

**Definition 4** ***(Tree isomorphism - rooted, labeled, unordered trees)*** *Let tree $T_1$ be defined by a set of vertices $V_1$ and a set of edges $E_1$. Another tree $T_2$ with vertices $V_2$ and edges $E_2$ is isomorphic to $T_1$ if and only if there exists a bijection function $f : V_1 \to V_2$ such that:*
*- $f(x)$ and $f(y)$ are adjacent if and only if $x$ is adjacent to $y$.*
*- The root node is conserved, i.e., denoting the root of the first tree as $r_1 \in V_1$ and the second tree as $r_2 \in V_2$, $f(r_1) = r_2$.*
*- All pairs of vertices matched by the function $f$ (i.e., all pairs $x \in V_1$ and $f(x) \in V_2$) share the same labels.*

**Definition 5** ***($k$-anonymity on hierarchical data)*** *A set of trees $T_1, T_2, ..., T_k$ form a $k$-anonymous group, if and only if all possible pairs $(T_i, T_j)$ where $T_i \in T_1, T_2, ..., T_k$ and $T_j \in T_1, T_2, ..., T_k$ are pairwise isomorphic. A forest (collection of trees) $F$ is $k$-anonymous, if all trees in $F$ belong to some $k$-anonymous group.*

2-anonymous versions of the two trees in Fig. 2.4 are given in Fig. 2.5. There are multiple ways of transforming these trees to fit the definition of $k$-anonymity, and the one shown in Fig. 2.5 is just one of them. The quality of these anonymizations depend mainly on how much data utility is lost (according to a given cost metric). In other words, an anonymization that fits the requirements of $k$-anonymity and yields the lowest utility loss is most desirable. For example, one can delete the whole branch that has course *CS13** in both trees, and the result would still be $k$-anonymous. However, that would result in higher data loss.

During the anonymization procedure shown, the trees have undergone certain changes which can be analyzed under two categories: (1) Generalization and suppression of quasi-identifier values. These correspond to changes in node labels,

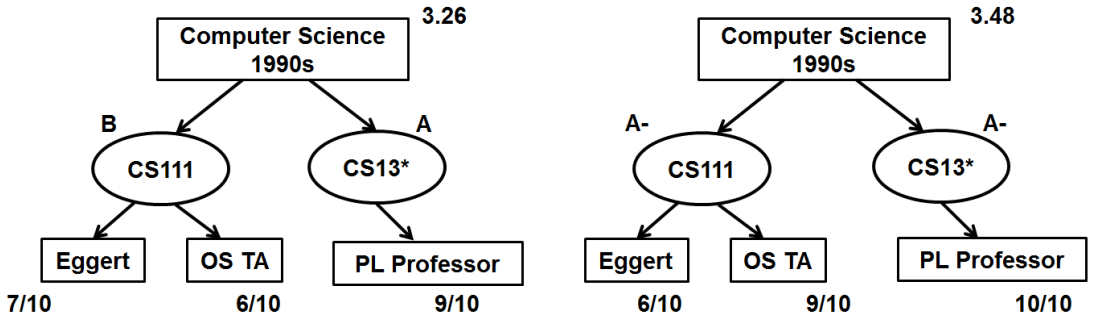



Figure 2.5: 2-anonymous trees.

e.g., *Millstein* and *Palsberg* have been generalized to *PL Professor*. (2) Removal of nodes and subtrees, e.g., the subtree rooted at *CS143* has been pruned. From a relational perspective, this translates to suppressing a tuple, including its join key. (If the join key is not suppressed, this would convey the information that the student has taken a class, but the name of that class is hidden.)

## 2.3 Rationale

Some obvious questions to ask are whether our definitions are needed and to what extent our scheme can preserve privacy. We investigate answers to these questions in this section. First, we will model an adversary that has partial knowledge of QIs in multiple relations. Notice that this is a different type of adversary compared to what is generally depicted in linkage attacks. This will help us demonstrate the need for cross-relational $k$-anonymity. Second, we will argue that single-table $k$-anonymity algorithms, as discussed in the literature, are not sufficient to ensure $k$-anonymity in hierarchical data. Third, we will consider the limitations of $k$-anonymity and suggest improvements.

**Observation 1** *An adversary may use cross-references and links between data nodes to violate individuals' privacy.*

Suppose that our adversary knows that a student is studying Computer Science, which allows him to deduce that this individual is one of the two in 2.4. The adversary also knows that this individual is taking CS132 (alternatively, the adversary knows he's taking CS143, or the adversary knows he's taking three classes). With this information, the adversary learns the GPA, class grades and evaluations given by this student. The adversary did not need full knowledge of QI values in one node/relation, but could use partial QI information from multiple nodes/relations to increase his chances (i.e., confidence) of singling out the target individual.

Let us consider a second adversary, one that knows the student was born in 1994 and is studying Computer Science. In this case, assume that the central relation has been properly 2-anonymized. Therefore the adversary can narrow potential candidates down to two trees, but cannot proceed further. However, if the remaining relations are not anonymized, the only extra information that the adversary needs is "Student has TA1 as his teaching assistant for CS111" or "Student is taking CS131" or "Student is taking CS143". (Any one of these three will be sufficient to distinguish the target individual.) So, just anonymizing one relation and not dealing with the remaining ones does not protect privacy, either.

**Observation 2** *In a hierarchical [multi-relational] database, anonymizing each level [relation] independently fails to protect privacy.*

**Observation 3** *Flattening hierarchical data into one giant relation and then running single-table k-anonymity algorithms on the resulting relation fails to protect privacy.*

For a detailed discussion on observations 2 and 3, the reader is referred to [37], which shows that single table anonymization algorithms cannot acknowledge *owners* of tuples (i.e., parent-child relationships). Also, each individual may have

an unbounded number of entries in a flattened table. Hence, anonymizations are not consistent between multiple relations. For example, consider the data in Fig. 2.3. In the first table, $S1$ and $S2$ are 2-anonymized. In the second table, though, the following anonymization is perfectly valid from the point of view of a single-table anonymization algorithm: $SG1$ and $SG2$ are 2-anonymized, $SG3$ and $SG4$ are 2-anonymized, $SG5$ is suppressed. However, it is obvious to see that this does not protect individuals' privacy: An adversary learns $S1$ took 3 classes, higher than the number of classes $S2$ took.

The papers cited above also illustrate that converting the database to an appropriate bitmap, or generalizing/suppressing join keys, or considering join keys as sensitive attributes (or quasi-identifiers) are all undesirable; since these methods either overly reduce data utility or cannot provide satisfactory privacy. We therefore conclude that it is necessary to employ the definitions of $k$-anonymity that we have given and provide solutions applicable to this new problem.

**Observation 4** *$k$-anonymity does not consider the distribution of sensitive values, and hence fails to conceal sensitive information that lacks diversity.*

In a $k$-anonymous group of trees, if the sensitive value of a certain node is the same in all $k$ trees, then an adversary can learn this sensitive value simply by looking at the group and not singling out the individual. For example, both students in Fig. 2.4 might have received the same letter grade in *CS111*. There are certain ways to aid this problem, such as the use of $l$-diversity on trees. However, we should note that such definitions are stricter than $k$-anonymity, and usually result in higher data loss.

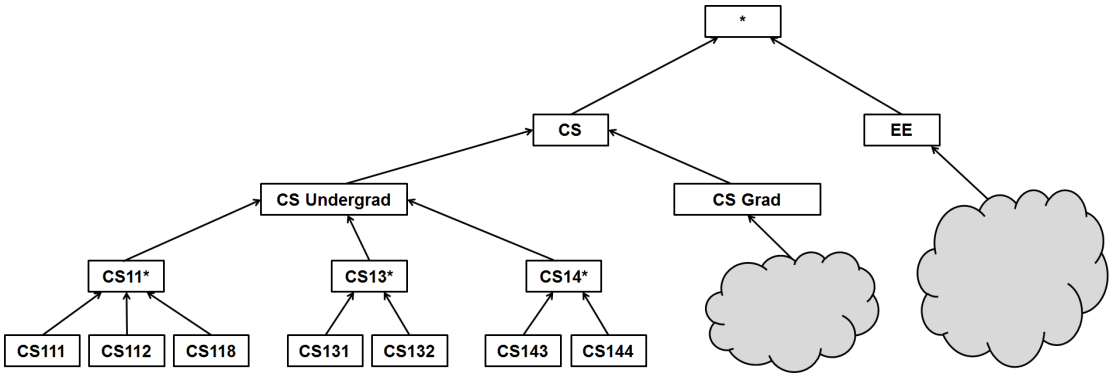



Figure 2.6: Part of a domain generalization hierarchy for courses.

## 2.4 Assumptions

### 2.4.1 Generalization Hierarchies

As stated earlier, we achieve anonymity using generalizations and suppressions on QI values. In case of generalizations, we make use of pre-defined *domain generalization hierarchies (DGH)*. We assume that a tree-structured DGH is available for each categorical attribute. Having such a taxonomy of values has been widely adopted in the literature [28][29][19][18].

Fig. 2.6 contains a sample DGH for university courses. Leaves of the DGH should contain all possible values that can be observed in the input (i.e., all courses offered at the university). The best (i.e., least costly) generalization of two values in the DGH is often their most recent common ancestor, e.g., it is most suitable to generalize *CS111* and *CS118* as *CS11**, whereas *CS111* and *CS143* can be generalized as *CS Undergrad* according to this DGH.

For numeric or continuous attributes (defined on a range), we say that either a pre-specified domain generalization hierarchy is available or a generalization scheme is easily inferable from the data. Fig. 2.7 provides a DGH for numerical attribute *year of birth*.

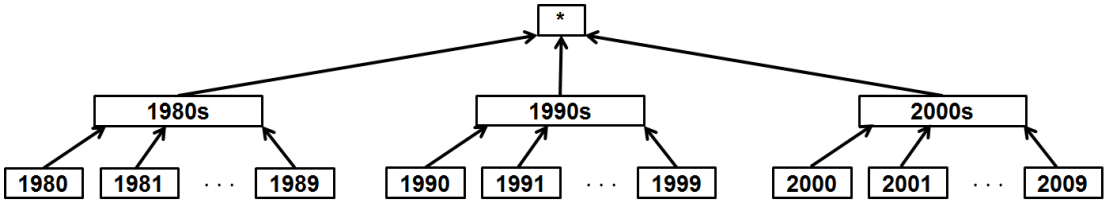



Figure 2.7: Part of a domain generalization hierarchy for years of birth.

### 2.4.2 Information/Utility Loss Metrics

Generalizations and value suppressions both cause loss of data utility, as specific values are either replaced with more general ones or completely obscured. Various metrics were proposed and used in relevant literature to calculate costs of these operations. LM [22], DM [5] and CM [22] are some examples. The *Classification Metric* (CM) is suitable when the purpose of anonymization is to train a classifier [19]. The *Discernibility Metric* (DM) penalizes tuples only according to the size of equivalence classes they are in, i.e., how many tuples (in our case, entitities/individuals) are indistinguishable from it. We will therefore use an adaptation of the *Generalized Loss Metric* (LM) in our examples and evaluation (also known as the *InfoLoss* or *VInfoLoss* metric in some sources, e.g., [8]). Our algorithms, however, are not dependent on the cost metric used, i.e., a user may define a new, monotonic cost metric that can be plugged in to our solution. (Monotonicity is required so that "more general" values that will appear in the final output are always penalized more than less general ones. For example, for years of birth, generalizing *1980s* and *1991* to * should receive higher penalty than generalizing *1991* and *1993* to *1990s*.)

Let us first define the LM cost of individual data values. Let $T$ be the DGH of a QI, and let $M$ denote the number of total leaf nodes in $T$. Let *val* be the value observed for that QI in our data tuple, and let $M_{val}$ denote the number of leaf nodes in the subtree rooted at value *val* within the DGH. Then, the LM cost of *val* is: $LM(val) = (M_{val} - 1)/(M - 1)$. For example, the LM cost of *CS11** in

Fig. 2.6 is *(3-1)/(# of total classes offered - 1)*. Leaves of a DGH have LM costs of zero, the root of a DGH has LM cost equal to one, and all remaining nodes have costs between 0 and 1.

We now extend this definition to hierarchical data. Let $f(val)$ be a function that retrieves the LM cost of an individual value $val$ as defined above. Let $F^*$ be an arbitrary forest that was subject to anonymization. Let $R$ denote a tree, $N$ denote a node in a tree, $|N|$ denote the number of QIs in $N$, and $N[k]$ denote the value of the $k$'th QI in $N$. The LM cost of $F^*$:

$$LM(F^*) = \frac{\sum_{R \in F^*} \sum_{N \in R} \sum_{i=1}^{|N|} f(N[i])}{\sum_{R \in F^*} \sum_{N \in R} |N|} \tag{2.1}$$

This definition is produced based on our tree representation. For the same definition expressed in multi-relational terms, refer to [37]. One can verify that the LM cost of a forest is normalized to a value in the range [0,1]. Deleted (pruned) nodes are considered to be fully suppressed (i.e., all their QIs are replaced by *), and hence receive the maximum penalty possible per node (which is 1).

# CHAPTER 3

# Anonymity in Hierarchical Data

In this chapter we examine a utility cost-aware function that can 2-anonymize two input trees, and then construct a clustering-based algorithm based on this function that allows generation of $k$-anonymous groups of trees for arbitrary values of $k$. We first outline some further notation and helper functions below.

## 3.1 Preliminaries

Let $a$ denote a tree node that consists of 0 or more QIs and sensitive attributes. A node may have an arbitrary number of children, which we denote by $a_1, a_2, ..., a_n$. Furthermore, let $tree(a)$ denote the tree rooted at $a$. In case of the cardinality symbol, assume that when it's used on tree nodes (e.g., $|a|$) it returns the number of children $a$ has. In all other cases (e.g., used on sets, lists etc.) it returns the number of elements in that set/list.

The examples given in Fig. 2.4 and Fig. 2.5 contain one "type" of node per level (for simplicity), e.g., nodes at depth 1 are all classes taken by the student, nodes at depth 2 are all evaluations given by the student. In a more general case, each level may contain nodes with different schema (in a multi-relational setting, this would refer to tuples in different relations), e.g., as in Fig. 2.2, nodes at depth 2 could also be books bought by the student. We have to account for cases in which multiple "types" of nodes co-exist at each level. It is often inappropriate to anonymize such nodes with one another, since they will most likely have different

type and number of QIs. This is addressed by the following function:

**compatible**($a$,$b$): Returns true if two tree nodes ($a$ and $b$) are schematically compatible to be anonymized (i.e., if database is multi-relational, originating from the same relation), false otherwise.

Next, we need to support generalization of compatible tree nodes. The following function takes as inputs two compatible nodes and transforms them so that they will be QI-wise indistinguishable.

**generalize**($a$,$b$): Given two tree nodes $a$ and $b$, finds and performs the lowest cost generalization of $a$ and $b$'s QIs according to their DGH and a cost metric (e.g., LM).

Not only the function described above, but also the anonymization procedure presented later will perform certain changes to trees (e.g., replacing QI values, deleting nodes/subtrees etc.). We need a way to calculate the cost of such operations, i.e., How much utility is lost while converting a tree to its anonymized version, according to a given cost metric?

**calculateCost**($tree(a)$,$tree(a')$): Given a tree rooted at $a$ and its modified version $a'$, calculates the cost incurred by the modifications.

Finally, we introduce a function that allows *deep copying* trees. A deep copy is needed whenever we need to change a tree in certain ways but would not like to apply these changes to the original tree yet.

**copyTree**($a$): Given a tree node $a$, deep copies the tree rooted at $a$ and returns the copy.

## 3.2 Anonymizing a Pair of Trees

Converting two arbitrary trees to a 2-anonymous pair is pivotal due to two reasons: First, we have to support such a procedure so that our overall solution can produce $k$-anonymous groups, as we intend to use this function as the main building block

for the rest of our algorithm. Second, we propose to use this function also as an indication of how *similar* two trees are, i.e., the "distance" between trees will be calculated as the cost of making them anonymous. If very little information would be lost by anonymizing a certain pair of trees, they are feasible candidates for one another, so the distance between them should be small.

In addition to the points above, given a fixed pair of trees as inputs, the function itself should be able to produce 2-anonymous versions of them with as little information loss as possible. Going back to Fig. 2.4, the following anonymization would also be legitimate: *CS111* in the first tree is anonymized with *CS132* in the second tree and *CS143* in the first tree is anonymized with *CS111* in the second tree (their children are also taken care of accordingly). However, the cost of this anonymization would be higher, since both of the nodes that are matched have to be generalized as *CS Undergrad*, as determined by their DGH in Fig. 2.6. In that regard, the choice of pairwise matching at each level has huge impact on the accuracy and information loss implied by the overall procedure.

We present a top-down anonymization function in Algorithm 1 and call this function **anonymize**. It can be studied in four steps; the starting point of each step is labeled on the algorithm. Step 1 checks whether the root nodes can be anonymized. If they can, they are matched and their QIs are generalized. If they are incompatible, their trees will be pruned. In step 2, the algorithm checks if further calculation is needed: Pairings and recursive steps need only be performed if both trees have children to be matched. Step 3 attempts to find the lowest-cost pairing between the two nodes' children. It does so using a look-ahead approach: It makes recursive calls to figure out "what would happen if these two children were actually paired and anonymized together". Notice that this requires several recursive calls on the deep copies of children. This implies that a suitable match is calculated not only by the QI values in the current level, but also taking into account the deeper levels; i.e., children are treated as subtrees and matched with

**Algorithm 1** Top-down anonymization

**Input:** Two trees rooted at $a$ and $b$ respectively
**Require:** $|a| \leq |b|$. Otherwise $a$ and $b$ can be interchanged as the first step within the procedure.

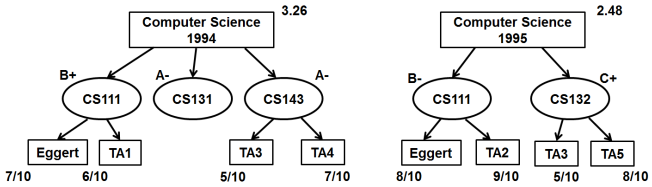
**procedure** ANONYMIZE(tree($a$),tree($b$))
  **if** $\neg$ compatible($a$,$b$) **then** ▷ Step 1
    delete tree($a$), tree($b$)
    **return**
  generalize($a$,$b$)
  **if** $|a| > 0$ **and** $|b| = 0$ **then** ▷ Step 2
    delete tree($a_1$), tree($a_2$), .., tree($a_n$)
    **return**
  **else if** $|a| = 0$ **and** $|b| > 0$ **then**
    delete tree($b_1$), tree($b_2$), .., tree($b_n$)
    **return**
  **else if** $|a| = 0$ **and** $|b| = 0$ **then**
    **return**
  pairs $\leftarrow$ [] ▷ Step 3
  **for** $i = 1$ to $|a|$ **do**
    minCost $\leftarrow +\infty$
    pairedIndex $\leftarrow \phi$ ▷ some special character
    **for** $j = 1$ to $|b|$ **do**
      **if** $j \in$ pairs **or** $\neg$ compatible($a_i$,$b_j$) **then**
        **continue** ▷ skip current iteration of inner loop
      $a'_i \leftarrow$ copyTree($a_i$)
      $b'_j \leftarrow$ copyTree($b_j$)
      anonymize(tree($a'_i$),tree($b'_j$))
      cost $\leftarrow$ calculateCost(tree($a_i$),tree($a'_i$))
      cost $\leftarrow$ cost + calculateCost(tree($b_j$),tree($b'_j$))
      **if** cost $<$ minCost **then**
        minCost $\leftarrow$ cost
        pairedIndex $\leftarrow j$
    pairs.append(pairedIndex)
  **for** $i = 1$ to $|a|$ **do** ▷ Step 4
    **if** pairs[i] $\neq \phi$ **then**
      anonymize($a_i$,$b_{pairs[i]}$)
  **for** $i = 1$ to $|a|$ **do**
    **if** pairs[i] $= \phi$ **then**
      delete tree($a_i$)
  **for** $j = 1$ to $|b|$ **do**
    **if** $j \notin$ pairs **then**
      delete tree($b_j$)
  **return**

appropriate subtrees (as opposed to being treated as nodes and matched with appropriate nodes). Also, the pairing here is done greedily, as explained in the next paragraph. Finally, in step 4, the best pairing found in the previous step is applied to the trees. Recursive calls are made to dive into each pair of children that are matched. Children for which no suitable pair could be found will be pruned in the clean-up phase (lines 34-39). A sample run is given in Fig. 3.1.

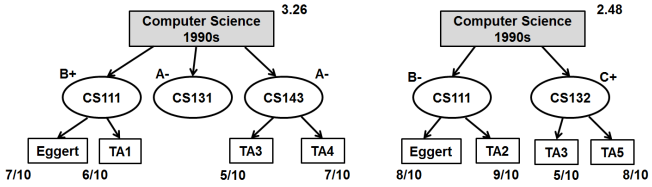
The choice of greediness in step 3 comes from the fact that the complexity of finding all pairings between two sets is high. This becomes a significant problem when the branching factor of input trees are large. Let $a$ have $n$ children and $b$ have $m$ children, where $m \geq n$. The total number of possible pairings between their children is (as determined by the total number of permutations): $\binom{m}{n} * n!$, which implies exponential complexity. We reduce this to quadratic complexity ($O(m^2)$): $a_1$ is paired with one of $b$'s children, then $a_2$ is paired with one of the remaining unmatched children of $b$ etc. Although we do not formally present results for this greedy versus optimal search cases, we did implement and test both. Even with our small datasets with average branching factors of 6-7, clustering with greedy search took $< 5$ minutes, whereas clustering with optimal search took several hours to complete. We therefore concluded that, for any practical database, optimal search with exponential complexity is infeasible.

## 3.3 Clustering for $k$-Anonymity

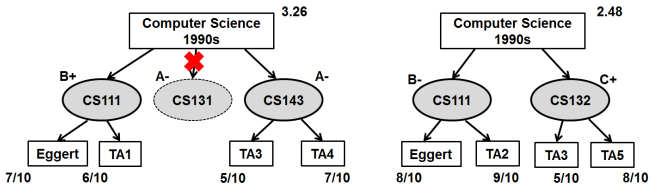
We designed a clustering algorithm based on the notion of agglomerative clustering to create groups of $k$-anonymous trees for $k > 2$. Other clustering approaches are also viable, such as the application of various well-known techniques (e.g., k-means) or the use of clustering methods that are created specifically for data privacy, e.g., [1], [3] (although most of such algorithms are only evaluated on tabular or single-relational data). Our main design concern was to create an



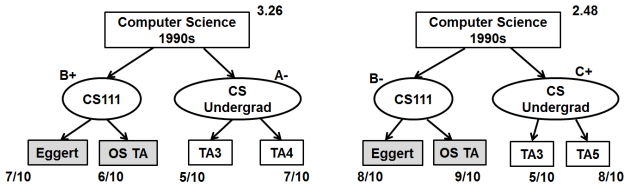

(a) The algorithm takes two trees as inputs.



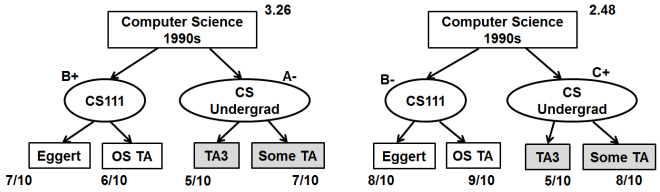

(b) First, root nodes are matched and generalized.




(c) Next, the algorithm finds a suitable pairing between nodes at depth 1. Notice that CS132 in the second tree is matched with CS143 instead of CS131 in the first tree, although CS131 looks like a more lucrative candidate based on its label. The algorithm recognizes that matching CS131 with CS132 would result in higher data loss in the next step, since CS132's children would have to be pruned.




(d) After the pairing is complete, each pair is handled independently.




(e) Once the two subtrees containing CS111 are taken care of, the algorithm moves on to the next pair (CS143-CS132).

Figure 3.1: Algorithm 1 anonymizing two trees step by step, in a top-down manner

algorithm that does not rely on any other parameter than $k$.

**Algorithm 2** Clustering procedure

**Input:** A forest of $n$ trees ($t_1$, $t_2$, .., $t_n$) and parameter $k$

```
1:  procedure ClusterTrees
2:      Initialize n clusters, such that the cluster representative of each c_i is t_i
3:      Build initial n × n distance matrix dist
4:      for i = 1 to n do
5:          for j = 1 to n do
6:              if i ≤ j then
7:                  dist[i][j] = +∞
8:              else
9:                  Copy cluster representatives
10:                 Use anonymize on these copies
11:                 Calculate cost of anonymization, i.e., distance between c_i and c_j
12:                 Insert cost to dist[i][j]
13:     while multiple active clusters exist do
14:         Find c_i and c_j that have minimum distance in dist
15:         Merge c_i and c_j using anonymize
16:         Update cluster representatives
17:         if resulting cluster has ≥ k trees then
18:             Remove cluster from distance matrix
19:         else
20:             Update distance matrix
21:     if there is a left-over (residual) cluster then
22:         Delete trees within that cluster
```

Algorithm 2 provides a high-level description of our clustering procedure. We use the concept of cluster representatives: Each cluster has a representative tree that summarizes all the trees within the cluster. Distance between two clusters is found by computing the distance between their representatives. When two clusters are merged, updating their representatives is a simple call to the **anonymize** function. Individual trees (i.e., a cluster of size $p$ will contain $p$ trees from the initial input) can be kept with or without updates. (Updating them on the fly will require multiple additional pairwise calls to *anonymize* and therefore might be undesirable.)

Notice that multiple calls to the **anonymize** function will be needed within this algorithm: (1) while building the initial distance matrix, (2) merging clusters

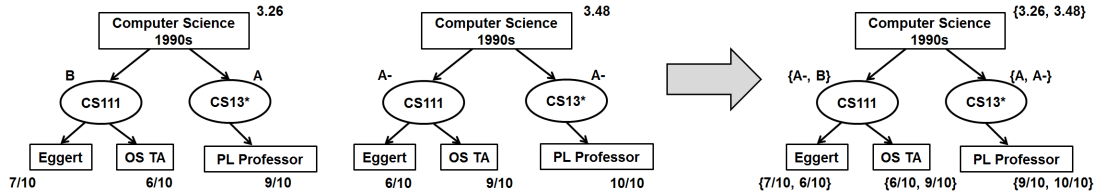



Figure 3.2: 2-anonymous trees in a cluster and their cluster representative.

and updating cluster representatives, and (3) updating the distance matrix after two elements are clustered (i.e., re-calculating distances from the resulting cluster to other active clusters).

We draw a sample cluster representative that would be generated if the two trees in Fig. 2.5 were placed in the same cluster. This process is depicted in Fig. 3.2. Since $k$-anonymity requires indistinguishability with respect to QIs, the cluster representative shares exactly the same QIs as the (anonymized versions of the) trees within its cluster. Adjacency information of nodes is also unchanged. Sensitive attributes of matching nodes (implied by the pairwise mapping of **anonymize** while children are paired with each other) are collected together as a set. For example, the first student received *A-* from *CS111* and the other student received *B*, so their cluster representative summarizes this in a collection/set {*A-, B*}. We should clarify that this is treated as a set of values for one sensitive attribute, rather than multiple sensitive attributes. A node containing $m$ sensitive attributes should have $m$ such sets. Since a cluster representative itself is essentially a tree, the functions and procedures described in previous sections require no or trivial modifications to work on cluster representatives.

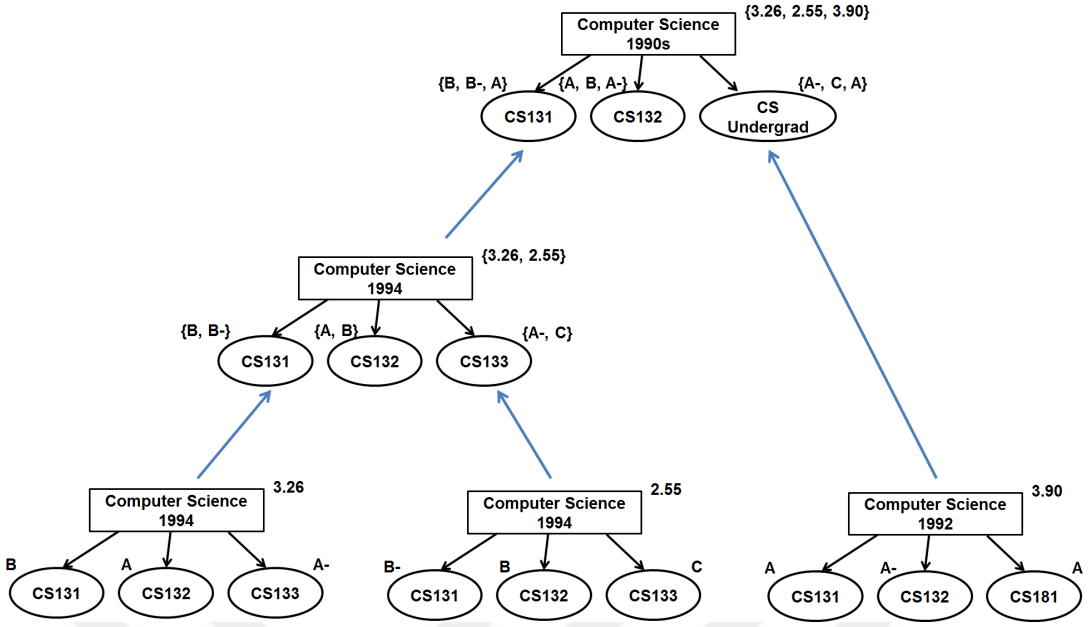



Figure 3.3: Sample run for Algorithm 2, with 3 trees. The leftmost two trees are clustered first, and the third tree joins afterwards. Intermediate and final cluster representatives are drawn above.

## 3.4 Experiments

### 3.4.1 Experimental Setup and Datasets

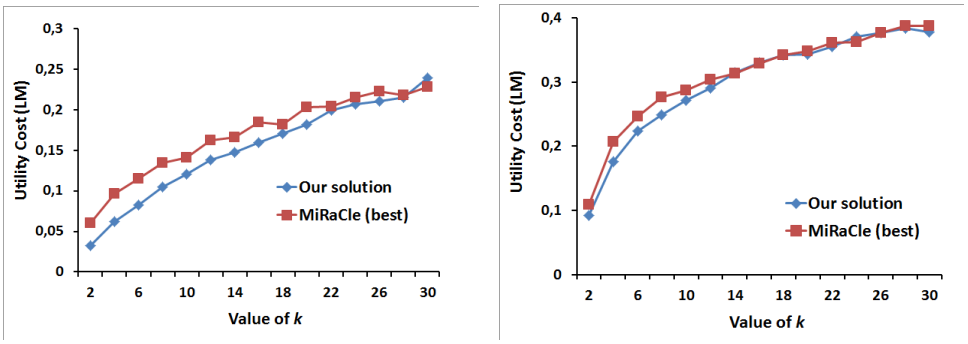
We generated synthetic datasets to test the utility and speed of our algorithms. We use a schema similar to the one in Fig. 2.2. Our data consists of students from Sabanci University's Computer Science (CS) program. We obtained actual data that contains the GPA and (partial) individual course grade information of 30 students from this year's graduating class. We simulated remaining students based on this sample, with the guidelines explained in the next paragraph. Algorithms were implemented in Java, and all experiments were conducted on a quad-core 2.40 GHz machine with 12GB main memory. Our setup was running 64-bit Windows 8 operating system and Java v8 u25. Comparisons are made in terms of execution time and information loss. We use the LM metric as defined earlier to quantify

how much data utility is lost, where lower cost implies preferable data sanitization.

We assumed that approximately the same number of students graduate every year, and set their year of birth 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. We randomly assigned a fixed (required) 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.

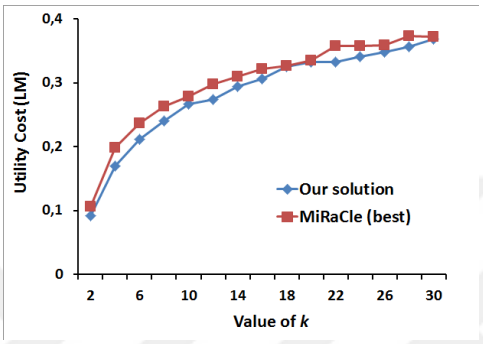
According to the university's graduation requirements, we calculated the triple ratio *(o:c:t)*, where $o$: number of obligatory classes, $c$: number of core classes, and $t$: average number of technical electives. To test with different sizes of datasets with varying branching factor, we reduced the number of classes a student would take, while keeping the ratio *(o:c:t)* constant. The pool of available classes was also downsized accordingly (for each category). As a result, we report results on three datasets: small, medium and large. The small dataset contains $\sim$7000 tuples, the medium dataset contains $\sim$14000 tuples and the large dataset contains $\sim$19000 tuples.

Proper DGHs were created for each QI. Birth years were generalized as in Fig. 2.7. Courses were first generalized according to their level and then according to department, e.g., *CS301* $\rightarrow$ *CS3*** (CS third year) $\rightarrow$ *CS*. Books were generalized according to the type and level of class they were affiliated with.




(a) Small dataset

(b) Medium dataset




(c) Large dataset

Figure 3.4: Utility costs for increasing values of $k$

### 3.4.2 Results and Comparison

We implemented the state-of-the-art multi-relational $k$-anonymity presented in [37]: *MiRaCle*. We run our datasets through the solution presented here and then compare our results to the cost of anonymizations generated by *MiRaCle*. During our implementation and tests, we assumed default parameters for the *MiRaCle* algorithm, as described by its authors. There is a threshold parameter *th* in *MiRaCle*, and the authors experiment with different *th* (in the range [0-1]) to find the most suitable value for their experimental dataset. We repeated our experiments with various *th* ranging from 0 to 1, with increments of 0.1. While comparing *MiRaCle* with our algorithm, we picked the *th* value that led to the best anonymization (i.e., lowest data utility loss).

We summarize our results in Fig. 3.4. We should clarify that we picked the

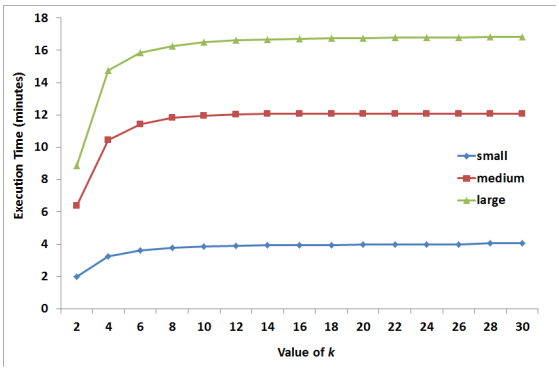



Figure 3.5: Execution time for the three experimental datasets.

best *th* and anonymization for all values of $k$ and datasets independent of other $k$s (as opposed to a fixed *th* for all $k$). We denote these "best" values we got from *MiRaCle* as *MiRaCle (best)*. Our graphs compare *MiRaCle* with our solution for varying values of $k$ from 2 to 30. As expected, as the $k$ value increases, costs of anonymization are also higher. This is simply because satisfying larger $k$ offers increased privacy, but is also more demanding to satisfy.

We see that our solution produces better $k$-anonymizations for the vast majority of $k$ values in the small and large datasets. Results are tied overall in the medium-sized dataset, with our algorithm outperforming *MiRaCle* for some values of $k$, and vice versa for others. We think that a particular strength that we have is that our solution is not dependent on any parameters. On the other hand, *MiRaCle*'s performance relies on a "good" choice of *th*. We observed that in the worst case *MiRaCle* can produce an output that has abruptly high utility loss (e.g., 7-8 times higher than its best case anonymization) and in the average case its output is 2-3 times more costly than its best case. These are both often significantly worse compared to our solution. Furthermore, a user might have to run *MiRaCle* numerous times with different *th* to get the best anonymization, but this might be undesirable since the process of anonymization can take several hours on large datasets, and re-running experiments might cause unnecessary loss of time and resources.

Finally, we report results regarding the efficiency of our solution. As Fig. 3.5 implies, our algorithms can complete within a reasonable amount of time on commodity hardware, running on data with several thousand tuples. We see that our solution is faster with smaller values of $k$. There is an observable increase in execution time as $k$ moves from 2 to 8, but it stabilizes from that point onwards. It should also be noted that the volume of data is a significant factor in efficiency, as anonymizing the medium dataset takes approximately 3 times the time it takes to anonymize the small dataset. Likewise, anonymizing the large dataset is $\sim$4 times more time-consuming. Nevertheless, anonymizing and publishing a dataset is a one-time cost. One can argue that this process can be done overnight, and has no bearing on a real-time system. Therefore, efficiency might not be the most crucial issue, and slower but more accurate algorithms might be worth employing. There is also the chance of using multiple independent machines: Say that UCLA wants to release data regarding all students. It's reasonable to assume that students from its CS program will have significantly different entries (in terms of classes taken etc.) than students from a social sciences program. Hence, it should be possible to divide data into groups of relevance and then run the clustering algorithm on each group to anonymize them independently.

# CHAPTER 4

# Diversity in Hierarchical Data

As noted in Chapter 2, $k$-anonymity is a privacy definition that ignores sensitive attributes and therefore it is susceptible to attacks when all trees in a QI-wise indistinguishable group share the same sensitive value. Or, if 80% of that group have the same sensitive value, an adversary can be 80% confident regarding the sensitive value of an individual that he determined is within that group. Another type of attack is the **background knowledge attack** [31]: Say that an adversary knows that a friend of his has applied to grad school, and this friend also told him that he was rejected because his GPA was not good enough. Equipped with this information, the adversary starts hunting for his friend's records in a 4-anonymous education dataset published by his friend's undergrad institution. Using QI values, the adversary links his friend to one of 4 records in the database, but cannot proceed further. However, the adversary sees that 3 of the students in that group have GPA $> 3.8$. Then, he can be fairly confident that his friend's data record is the 4th (remaining) one. The privacy leak in this case occurs due to a lack of diversity in sensitive values. If GPA scores were more evenly distributed within anonymized groups (i.e., the GPA values of the four students had more variety) then it would have been possible to avoid this problem. The notion of $l$-diversity formalizes and addresses this issue.

## 4.1 Defining Diversity

We need to first clarify what it means for a set of observed values to be "diverse". In line with existing literature and basic notions, we define the following **diversity requirement**: A sensitive attribute is either categorical or continuous (on a range). For categorical attributes (e.g., letter grade), an $l$-diverse set must contain $l$ distinct values. For continuous attributes (e.g., GPA), the allowed range is divided into non-overlapping buckets (pre-defined by the user, does not have to be of equal size etc.), e.g., the GPA range [0.00-4.00] can be divided into 4 buckets: *[0.00-1.00], ..., (3.00-4.00].* An $l$-diverse set must contain values from $l$ different buckets, e.g., the values {*2.34, 3.20*} are 2-diverse but {*2.34, 2.36*} are not.

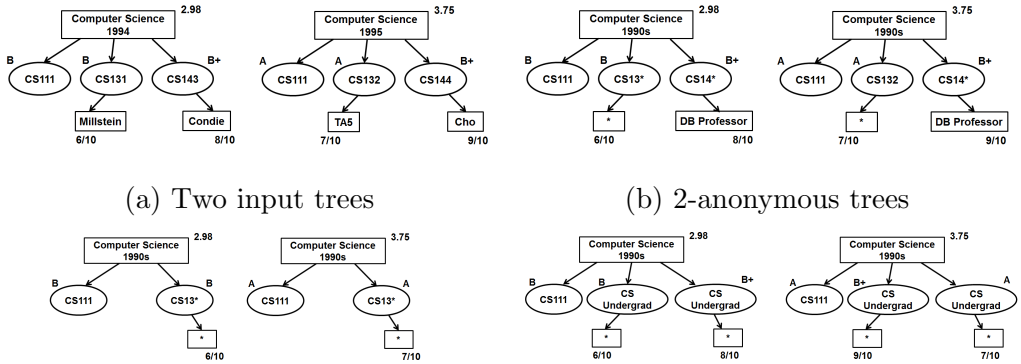
Note that there are other instantiations that take into account the probability of occurrence of a sensitive value in the dataset, e.g., *entropy $l$-diversity* and *recursive $(c, l)$-diversity.* These are usually considered to be more restrictive [31].

Recall Definition 4 and Definition 5 from Chapter 2: We determine anonymity (and isomorphism) between two trees using a bijection function that maps $T_1$'s vertices to $T_2$'s vertices. Each pair of vertices that are matched by this bijection function $f$ are required to share the same adjacency information and label. We extend this definition to support $l$-diversity as below.

**Definition 6** ***($l$-diversity in hierarchical data)*** *We say that trees $T_1, .., T_k$ form an $l$-diverse group, if:*

*- They are QI-wise indistinguishable: All possible pairs $(T_i, T_j)$, where $T_i \in T_1, .., T_k$ and $T_j \in T_1, .., T_k$, are pairwise isomorphic.*

*- Sensitive attributes of each corresponding node adhere to the diversity requirement: For $1 \leq i \leq k-1$, let $f_i$ be a bijection function that maps $T_1$'s vertices to $T_{i+1}$'s vertices, as in the definition of pairwise tree isomorphism. Let $T_1$'s vertices be labeled as $v_1^1, v_2^1, v_3^1, .., v_n^1$. Then, there should exist a set of functions $\{f_1, f_2, .., f_{k-1}\}$ such that $\forall x \in \{1, 2, .., n\}$, the set $\{v_x^1, f_1(v_x^1), f_2(v_x^1), .., f_{k-1}(v_x^1)\}$*




(a) Two input trees

(b) 2-anonymous trees

(c) 2-diverse trees

(d) 2-diverse trees

Figure 4.1: Multiple ways of diversifying two trees

*contains l sensitive values that satisfy the diversity requirement.*

A forest (collection of trees) $F$ is $l$-diverse if all of the trees inside it belong to some $l$-diverse group. There can be several $l$-diverse groups within $F$.

In Fig. 4.1 we see $l$-diversity in action. The pairwise matching of CS111 with CS111, CS131 with CS132, and CS143 with CS144 is perfectly legitimate from the point of view of $k$-anonymity (and is also the least costly). However, the 2-anonymous trees in Fig. 4.1b leak that both students received a B+ from CS144. We can use $l$-diversity to fix this issue, and there can be a couple of approaches to achieve that: (1) We take the 2-anonymous trees in Fig. 4.1b and we prune/delete every node and subtree that violates diversity. Fig. 4.1c would be the result. But this is often a rather drastic solution - what if all paired nodes, by consequence, have the same sensitive values? Should we have to prune all of them? (2) A more reasonable approach is depicted in Fig. 4.1d. In this case, we take into account the sensitive values of nodes while matching them, so that each pair that is matched will satisfy the diversity requirement. For example, CS111 is matched with CS111, CS131 is matched with CS144 and CS143 is matched with CS132. (We changed the order of nodes in Fig. 4.1d to demonstrate the matching.) This applies a relatively more costly matching compared to $k$-anonymity, but helps more nodes

"survive", therefore leading to lower information loss in the long run.

## 4.2 Incorporating Diversity into Our Solution

We extend the solution we have for $k$-anonymity to support $l$-diversity. Let us once again start by diversifying a pair of trees, and then develop a clustering algorithm on top of that.

To implement our observation concerning Fig. 4.1, we create a function called **isDiverse**. Given two tree nodes $a$ and $b$, **isDiverse($a$,$b$)** returns true if the sensitive values of $a$ and $b$ satisfy the diversity requirement, and false otherwise. This is straightforward when we think of $a$ and $b$ as tree nodes, but not that much when they are cluster representatives. Recall that we use cluster representatives to summarize the trees within a cluster, and the sensitive values of nodes are collected together as a set. So, how do we decide whether two cluster representatives are "sensitive value"wise diverse or not? For the purposes of the clustering procedure we discussed earlier, we demand that a cluster of size $p$ is always $p$-diverse. If a legitimate cluster of size $p_1$ and another legitimate cluster of size $p_2$ are merged (where $p = p_1 + p_2$) we'd like the resulting cluster to be $p$-diverse. This dictates the behavior of *isDiverse*. In other words, if we call the sensitive value set of the first node $S_1$ and the second node $S_2$, $S_1 \cap S_2 = \emptyset$ should hold.

Similar to **anonymize** that achieves top-down anonymization, we present **diversify** in Algorithm 3 for top-down diversification. **diversify** is technically just an extended version of **anonymize** that considers sensitive values during the pairwise matching phase. The remaining intuition stays the same.

We can directly employ **diversify** in the clustering procedure we explained in the previous chapter (simply replace *anonymize* by *diversify* in Algorithm 2). We call this approach the **naive clustering** technique. However, we design a second clustering algorithm which relaxes the previous requirement regarding a cluster

**Algorithm 3** Top-down diversification

**Input:** Two trees rooted at $a$ and $b$ respectively
**Require:** $|a| \leq |b|$. Otherwise $a$ and $b$ can be interchanged as the first step within the procedure.

1: **procedure** DIVERSIFY(tree($a$),tree($b$))
2: **if** ¬ compatible($a$,$b$) **or** ¬ isDiverse($a$,$b$) **then** ▷ Step 1
3: delete tree($a$), tree($b$)
4: **return**
5: generalize($a$,$b$)
6: **if** $|a| > 0$ **and** $|b| = 0$ **then** ▷ Step 2
7: delete tree($a_1$), tree($a_2$), .., tree($a_n$)
8: **return**
9: **else if** $|a| = 0$ **and** $|b| > 0$ **then**
10: delete tree($b_1$), tree($b_2$), .., tree($b_n$)
11: **return**
12: **else if** $|a| = 0$ **and** $|b| = 0$ **then**
13: **return**
14: pairs ← [] ▷ Step 3
15: **for** $i = 1$ to $|a|$ **do**
16: minCost ← $+\infty$
17: pairedIndex ← $\phi$ ▷ some special character
18: **for** $j = 1$ to $|b|$ **do**
19: **if** $j \in$ pairs **or** ¬ compatible($a_i$,$b_j$) **or** ¬ isDiverse($a_i$,$b_j$) **then**
20: **continue** ▷ skip current iteration of inner loop
21: $a'_i$ ← copyTree($a_i$)
22: $b'_j$ ← copyTree($b_j$)
23: diversify(tree($a'_i$),tree($b'_j$))
24: cost ← calculateCost(tree($a_i$),tree($a'_i$))
25: cost ← cost + calculateCost(tree($b_j$),tree($b'_j$))
26: **if** cost < minCost **then**
27: minCost ← cost
28: pairedIndex ← $j$
29: pairs.append(pairedIndex)
30: **for** $i = 1$ to $|a|$ **do** ▷ Step 4
31: **if** pairs[i] $\neq \phi$ **then**
32: diversify($a_i$,$b_{pairs[i]}$)
33: **for** $i = 1$ to $|a|$ **do**
34: **if** pairs[i] $= \phi$ **then**
35: delete tree($a_i$)
36: **for** $j = 1$ to $|b|$ **do**
37: **if** $j \notin$ pairs **then**
38: delete tree($b_j$)
39: **return**

of size $p$ always being $p$-diverse. Our main concern with the naive procedure was that stricter privacy requirements (i.e., higher values of $l$) might be harder to satisfy, simply because of the distribution of sensitive values (i.e., if there is a lack of diversity in sensitive values observed in the dataset).

**Algorithm 4** Extended clustering procedure

**Input:** A forest of $n$ trees ($t_1$, $t_2$, .., $t_n$), privacy parameter $l$, maximum cluster size parameter $\alpha$

```
1: procedure ClusteringExtended
2:     Initialize n clusters, such that the cluster representative of each c_i is t_i
3:     Set c_i as not anonymizeable
4:     Build initial n × n distance matrix dist
5:     for i = 1 to n do
6:         for j = 1 to n do
7:             if i ≤ j then
8:                 dist[i][j] = +∞
9:             else
10:                Copy cluster representatives
11:                Use diversify on these copies
12:                Calculate cost of anonymization, i.e., distance between c_i and c_j
13:                Insert cost to dist[i][j]
14:    while multiple active clusters exist do
15:        Find c_i and c_j that have minimum distance
16:        if c_i is anonymizeable or c_j is anonymizeable then
17:            Merge c_i and c_j using anonymize
18:        else
19:            Merge c_i and c_j using diversify
20:        Update cluster representatives
21:        if resulting cluster has ≥ l + α trees then
22:            Remove cluster from distance matrix
23:        else if resulting cluster has ≥ l trees then
24:            Set resulting cluster to be anonymizeable
25:            Update distance matrix with distances from this cluster to others,
   calculated using anonymize
26:        else
27:            Update distance matrix
28:    if there is a left-over (residual) cluster then
29:        Delete trees within that cluster
```

We present the new clustering procedure in Algorithm 4. The main idea here is that a cluster of size $\geq l$ already satisfies the diversity requirement, but it does not

have to be removed from the search space. Instead, it can "recruit" other clusters using *anonymize*. We control this behavior using the keyword *anonymizeable* - a cluster that already satisfies $l$-diversity is set to be anonymizeable, and from that point onwards it can recruit using **anonymize** rather than **diversify**. This allows generation of $k$-anonymous $l$-diverse clusters (where $k \geq l$), i.e., clusters of size (potentially) much larger than $l$, but still $l$-diverse. The advantageous part of this is that anonymity is always a less demanding notion of privacy compared to diversity. There might exist two nodes that would have to be pruned if they are diversified (due to lack of diversity in sensitive values) but might in fact be suitable to be anonymized.

The $\alpha$ parameter in the algorithm is used as a *stop condition*: Clusters must be removed from the search space at some point, otherwise they will keep merging until a single cluster is left. Notice that with $\alpha = 0$, this procedure becomes essentially the same as the naive clustering. We should also clarify how to update distances (line 27) in **extended clustering**: Whenever the algorithm reaches line 27, it means that the resulting cluster (from the latest merge operation) has $< l$ trees, and therefore does not satisfy $l$-diversity yet. Determining distance from it to another active cluster, say $c_j$, depends on whether $c_j$ is *anonymizeable* or not. If so, the distance should be calculated using *anonymize*, otherwise using *diversify*.

## 4.3 Experiments

### 4.3.1 Experimental Setup and Datasets

We use the the same experimental setup and datasets that are generated using the same principles as the previous section. For the sensitive attribute GPA, we divide its range [0.00-4.00] into 8 distinct buckets of size 0.50 (hence resulting in 8 possible sensitive values). 11 distinct values are used for letter grades, ranging

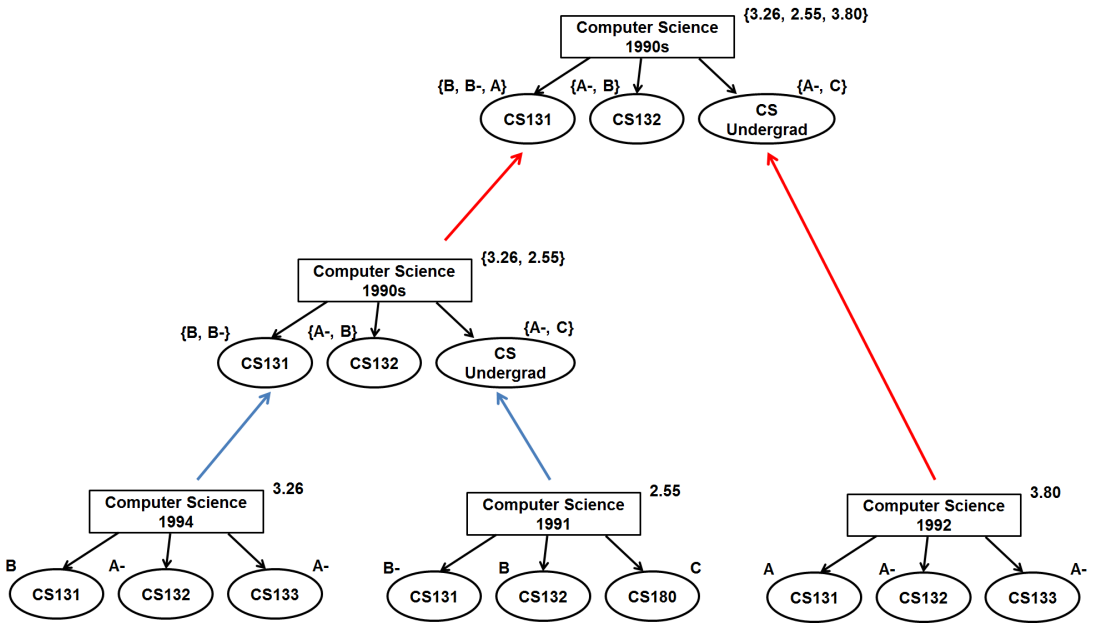



Figure 4.2: Sample run for Algorithm 4, with $l = 2$ and $\alpha > 0$. Although the first and the third trees are QI-wise better fit for one another, their sensitive values lack diversity. The algorithm therefore chooses to cluster the first two trees first, using diversify (marked with blue arrows). The extended clustering procedure allows the third tree to join (marked with red arrows) after $l = 2$ is satisfied.

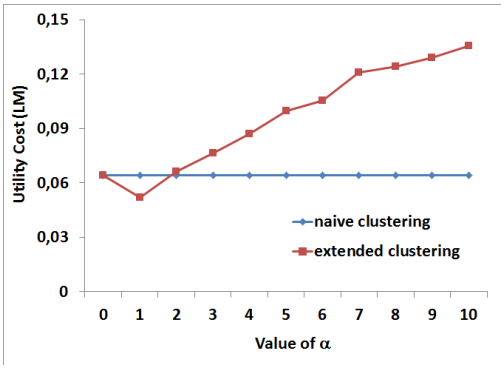
from A, A-, B+, .., F. Prices of books range between \$0 and \$80, and this range is also divided into 8 buckets of size \$10.

We would like to note that the first and third levels in our schema contain 8 distinct buckets, and hence we only experiment with $l$ ranging from 2 to 6. In fact, when $l = 6$, we will see that the LM costs become significantly large. This can be amended by increasing the number of buckets (e.g., for GPA values, have 16 buckets of size 0.25). We left it as is, in order to experiment with cases where there is significant utility loss and also emphasize the impact of the extended clustering algorithm in such cases. Another point worth considering is that we simulated GPA values using a normal distribution. If we used a uniform distribution instead, for instance, it would have been much easier to obtain a flat distribution of sensitive values (which would consequently mean higher number of candidates that can satisfy larger $l$). The choices we made, while generating the rules for our experiment data, aimed to achieve more realistic data that captures the characteristics of the sample we had.

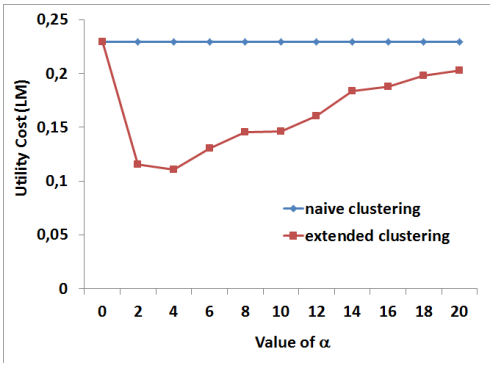
### 4.3.2 Effect of $l$ and $\alpha$

In Fig. 4.3, we graph the LM costs of anonymizing the dataset for different values of privacy parameter $l$. As expected, cost increases as $l$ is incremented from 2 to 6. The naive clustering procedure produces good results for small values of $l$, but as the privacy requirement gets more demanding (e.g., $l = 5$ or 6) there is higher incentive to use the extended version. We see that the choice of $\alpha$ is important to achieve better results. Utility costs make a U-shaped curve, where the drop is often much steeper than the rising portion. For high values of $\alpha$, the cost of using the extended version may surpass the naive version.

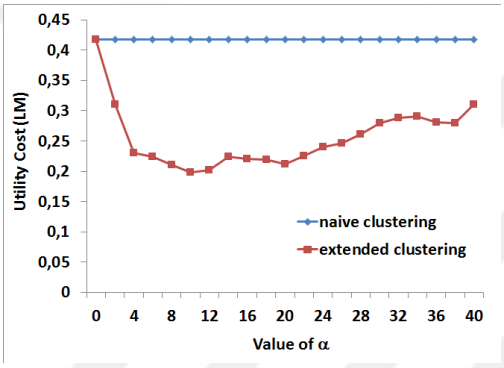
Different values of $\alpha$ seem to be better for different values of $l$, and optimal values of $\alpha$ are higher for increased values of $l$. These graphs were obtained using



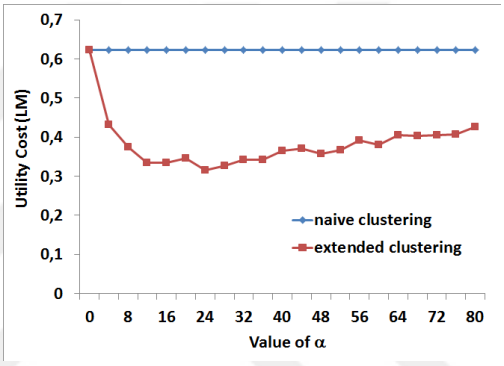

(a) $l = 2$



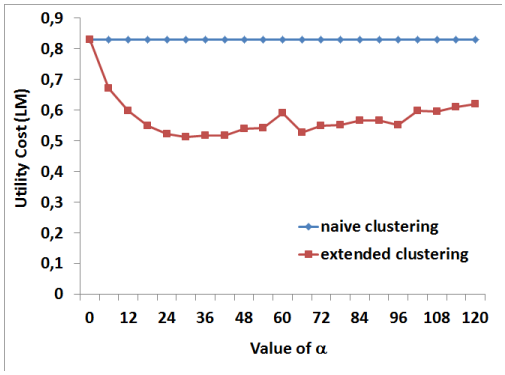

(b) $l = 3$




(c) $l = 4$

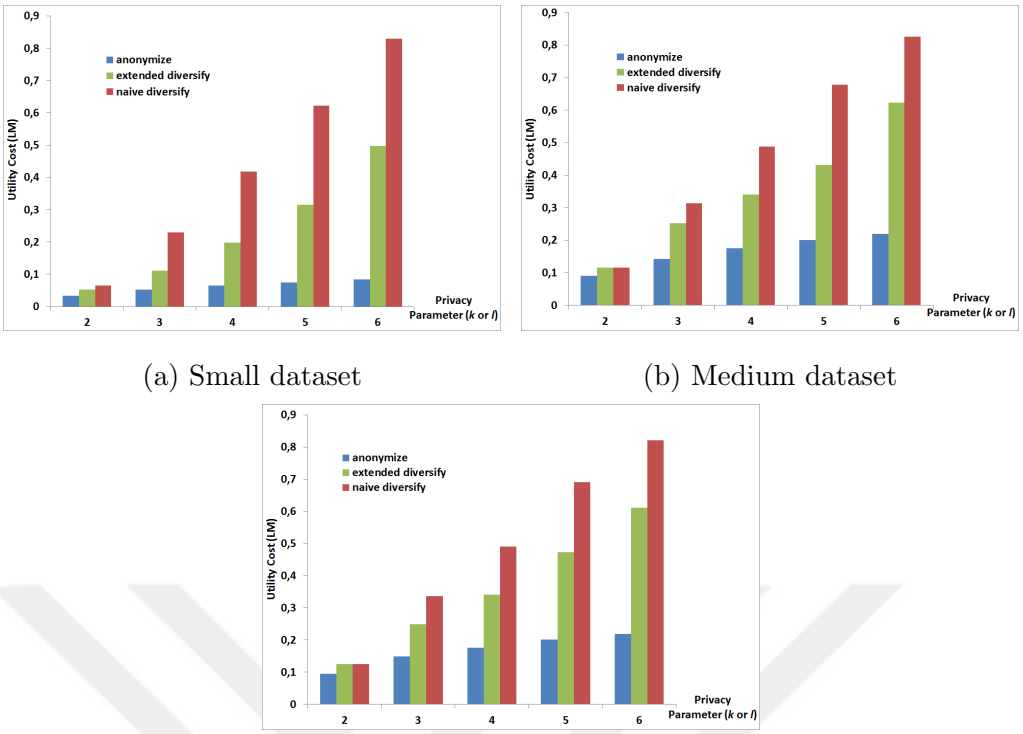



(d) $l = 5$




(e) $l = 6$

Figure 4.3: Change in utility cost for different values of $l$ and $\alpha$




(a) Small dataset

(b) Medium dataset

(c) Large dataset

Figure 4.4: Comparison between anonymity, naive and extended diversification

the small dataset. We would expect similar curves of utility loss for different datasets (although optimal values etc. might be somewhat different). We validated this claim using the medium and large datasets, but for brevity we do not include detailed results for them. In any case, when used with suitable $\alpha$, the extended algorithm is shown to be able to decrease the utility loss reported by the naive algorithm by half.

### 4.3.3 Comparison with Anonymity

An interesting question is how $l$-diversity compares to $k$-anonymity in the case of hierarchical data. We hope that this study offers some insight regarding the question: "What is the additional cost of applying a stricter notion of privacy?". Obviously, there is a trade-off between privacy requirements and data utility,

since demanding higher privacy would most likely cause more data records to be generalized or masked. We already investigated this issue from the point of view of increasing the privacy parameter ($k$ for anonymity and $l$ for diversity), but now we would like to offer a comparison between the different notions of privacy. Consequently, data owners may decide whether it is reasonable to employ one over the other, i.e., if it is worth using $l$-diversity instead of $k$-anonymity for the sake of better privacy protection.

We perform our tests on the three different sizes of datasets explained in the previous chapter. Our results are summarized in Fig. 4.4. For the *extended diversify* case, we experiment with different $\alpha$ and pick the value that offers the highest utility anonymization and report that value in our graphs. Again, in all cases, an increase in $k$ or $l$ yields higher utility loss. We can observe that for small $k/l$, it might actually be reasonable to use diversification, as there is not a big gap between the utility loss of $k$-anonymity and *extended diversify*. However, for higher $k/l$, even the extended clustering procedure (for $l$-diversity) yields utility losses 4-5 times higher than $k$-anonymity.

One can also observe that the benefit of using the extended procedure can be different for various datasets. For instance, the difference between the naive and extended diversification algorithms is very significant for the small dataset, but their discrepancy is somewhat less for the other two datasets. We think that this is because larger datasets had more courses available per student; and therefore when two trees are being diversified, for each course in the first tree there are more courses available to match it with in the second tree. This increases the chance of finding a sensitive value-wise diverse match for each course.

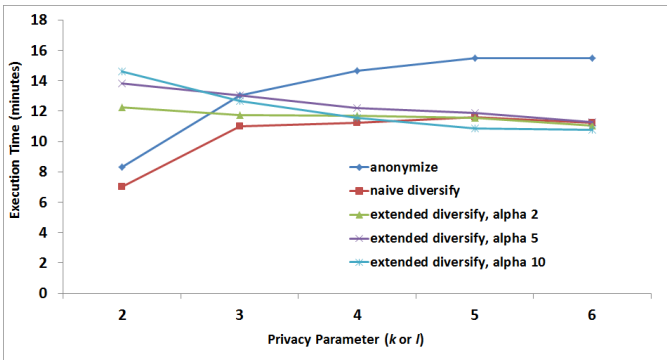



Figure 4.5: Execution times for the medium dataset

### 4.3.4 Efficiency

Similar to the results we obtained in the previous chapter, we see that both *anonymize* and *naive diversify* have higher execution time for increased values of $k$ and $l$, respectively. Also, Fig. 4.5 shows that *naive diversify* is faster than *anonymize*. This is because for some trees/subtrees, their sensitive values are not diverse when matched (also applies to root nodes, i.e., GPA). In that case, the algorithm does not need to proceed any further, which leads to less computation. For example, two students with GPA 3.72 and 3.73 cannot satisfy diversity, and hence the algorithm can terminate without having to match their classes etc.. Whereas in *anonymize*, there is no such early termination.

An interesting finding is that higher $\alpha$ leads to higher execution time for small $l$. Also, *extended diversify* often seems to be slower than *naive diversify* for small $l$. However, *anonymize* and *naive diversify* take longer to execute as $k/l$ gets larger. In contrast, the opposite is true for *extended diversify*: Its execution time is less for higher $l$. These cause cross-overs not only between *naive diversify* and *extended diversify*, but also between the *extended diversify* curves themselves (drawn for various $\alpha$).

# CHAPTER 5

# Towards Partial Publication

Let us take a step back and consider the assumptions we made so far. We model individuals' presence in a hierarchical database using a tree structure, and then our models offer anonymity (and diversity) among "individuals' trees". An important but somewhat implicit assumption we make here is that records in the output have to be explicitly linked to their owners (i.e., the outputs have to appear in tree structures as well), which allows no ambiguity regarding whom a particular tuple belongs to. In XML/JSON, this is achieved by object representations, e.g., we know that student $X$ has taken classes $Y_1, Y_2, ..$ and gave evalations $Z_1, Z_2, ...$ simply because records $Y_i$ and $Z_j$ appear under $X$. In multi-relational databases, this is achieved by the use of join keys across multiple relations, e.g.,in Fig. 2.5 we are certain that it is the first student who evaluated Prof. Eggert and gave a score of 7/10 because there is a join key that ties this evaluation record to the record that says this student took CS111, and another join key that ties the CS111 record to the central relation (i.e., root node). In order to preserve the structure of records, we allowed no generalizations of join keys, and suppressions only when the record had to be completely deleted.

In this chapter we relax this setting: What if the recipient of the data does not need to know about the "ownership" and links between records in multiple levels, and therefore the structure of the data can, in fact, be neglected? That is, what if join keys are irrelevant for the data recipient? E.g., the recipient would like to analyze the likelihood of students from different majors taking a certain class X,

but does not really care if class X and class Y were taken together by the same student. The requirements of this scenario are different than what has previously been discussed in the thesis.

We would like to demonstrate the feasibility of this new setting using relevant examples from the literature. The following single-relation schema is studied in [46]: *(patientID, occupation, zipcode, place of birth, test)*, where *test* is a sensitive attribute that contains health-related tests that patients took at a hospital. It is acknowledged that a patient may have several tests, and in that case multiple tuples in the relation may refer to the same individual. A credit card transaction database is presented in [53], with schema: *(age, occupation, postcode, vendor, amount)*, where {*age, occupation, postcode*} is the set of quasi-identifiers. In both of these cases, authors focus on anonymizing **tuples** rather than **individuals**, i.e., it is assumed that the data recipient will not be interested in knowing if (and which) tuples belong to the same patient/customer, how many transactions or hospital visits were made by an individual etc.. This is the opposite of what this thesis has discussed in the previous chapters. In particular, one may refer to Fig. 2.1 to see how a transaction schema would be modeled using our approach. But depending on the needs of the data recipient, the scenarios in [46] and [53] might as well be sufficient. This is the point of this chapter: How can we generalize their solutions and apply them to structurally rich, hierarchical data?

We should point out that $k$-anonymity and $l$-diversity of trees are still valid notions to achieve this (i.e., they would solve the problem we are posing) but they would be an overkill. Recall the process of matching nodes: While $T_1$ and $T_2$ are being anonymized, all nodes in $T_1$ are being matched with appropriate nodes in $T_2$ (ones that have no appropriate match are pruned). However, if there is no "sense of ownership", it should be viable to match a node in $T_1$ with a schematically compatible node in $T_2$, and another node in $T_1$ with a node in $T_3$.

## 5.1 Motivation

A particular case where all of this becomes interesting is what we call **partial publication**. In partial publication, we assume that the data owner has semi-structured data, i.e., multiple connected relations. There are potentially many data recipients that are interested in different portions of the data, e.g., data from only 2 of the relations out of a possible 4.

Let us first remind the reader of the relations we had in our education example (Fig. 2.2). Join keys (i.e., primary and foreign keys) are in **bold**, sensitive values are in *italics* and QIs are in regular font:

- $I_1$(**studentID**, major, YoB, *GPA*), which is the central relation.
- $I_2$(**studentID**, **geID**, class, *grade*)
- $I_3$(**geID**, instructor, *evaluation*)
- $I_4$(**geID**, book, *price*)

Consider the following motivating example: UCLA has collected data on its students in the form of the relations above, and would like to publish it to recipients $R_1$, $R_2$ and $R_3$. $R_1$ would like to mine the data to answer questions such as: "Do students from the engineering school give better evaluations for their instructors?" and "Do students with higher GPA give better evaluations?". It is sufficient for $R_1$ to receive data from $I_1$ and $I_3$. $R_1$ does not need any information regarding classes taken by students. $R_2$ would like to investigate if there are certain classes that people with high GPA prefer to take over other classes. Therefore $R_2$ only needs $I_1$ and $I_2$. $R_3$ would like to see how well students in the engineering field are doing compared to other majors, and hence only the central relation would suffice in his case. Not a single one of $R_1$, $R_2$ and $R_3$ needs to know about ownership or links between tuples, e.g., $R_1$ does not care if it was the same student who gave

all of the 10 worst evaluations in the dataset, or if it was 10 students with 1 bad evaluation each. (Whether $R_1$ **should** care about this is another issue of debate. If he does, we would ship him the $k$-anonymized trees.)

## 5.2 Definitions and Examples

Most of the material presented here presumes familiarity with the multi-relational snowflake schema setting discussed earlier. Therefore one should assume the validity of statements and definitions presented in Chapter 2, unless explicitly stated otherwise in this chapter.

**Definition 7** ***(Ancestor relation)*** *We say that $I_j$ is an ancestor relation of $I_k$, if and only if $I_k$ has a foreign key that is a primary key in $I_j$. We denote this relationship as $Anc(I_k) = I_j$.*

**Definition 8** ***(Ancestral path)*** *In a snowflake schema with central relation $CR$, the ancestral path of a relation $J$, denoted $AncPath(J)$, is an ordered set $\{I_1, I_2, ..., I_n\}$ where $I_1 = J$, $I_n = CR$ and $\forall I_i \in AncPath$, $Anc(I_i) = I_{i+1}$.*

Intuitively, an ancestral path is the (only) path from a relation to the central relation of the schema that follows primary and foreign keys. In our example, $AncPath(I_3) = \{I_3, I_2, I_1\}$ and $AncPath(I_1) = \{I_1\}$.

We denote a data recipient using the following transcript:
$Recipient(attr_1^{N_1}, attr_2^{N_2}, ..., attr_m^{N_m})$, where each $attr_i^{N_i}$ is a quasi-identifier or sensitive attribute in relation $N_i$ written as its superscript. Consider $R_1$, $R_2$ and $R_3$ described in the previous section:

- $R_1(major^{I_1}, YoB^{I_1}, GPA^{I_1}, instructor^{I_3}, evaluation^{I_3})$
- $R_2(major^{I_1}, YoB^{I_1}, GPA^{I_1}, class^{I_2}, grade^{I_2})$

- $R_3(major^{I_1}, YoB^{I_1}, GPA^{I_1})$

**Definition 9** ***(Relevant join)*** *We compute a relevant equi-join* ***RJ*** *concerning a recipient's transcript as follows. Let an intermediate product $L$ be computed as: $\forall attr^{N_i} \in transcript$, $L = \bigcup_{N_i} AncPath(N_i)$. Let $L^{'}$ contain the same elements as $L$, but with duplicates removed. The resulting $L^{'}$ will have the format $\{M_1, M_2, ..., M_k\}$, where each $M_j$ is a data relation from the input schema. Then, the relevant join is an equi-join of these relations using primary and foreign keys, i.e., $RJ = M_1 \bowtie M_2 \bowtie ... \bowtie M_k$.*

The relevant join for $R_1$ is $I_1 \bowtie I_2 \bowtie I_3$, for $R_2$ it is $I_1 \bowtie I_2$ and for $R_3$ it is $I_1$ only. In words, a relevant join tries to calculate a minimal join operation (i.e., consisting of minimum amount of input relations) that will generate an output relation that can be anonymized and shipped to the data recipient. A relevant join will: (1) always contain the central relation even in cases where no attributes in a recipient's transcript originate from the central relation, (2) contain all attributes in a recipient's list of desired attributes, i.e., transcript, (3) may contain several QI and sensitive attributes that are not required by the recipient (E.g., Consider a recipient $R_2^{'}$ that requires the following attributes: major, GPA, class, grade. A relevant join computed for this recipient will also have the attribute "YoB", which is not necessary for $R_2^{'}$.), and (4) all join keys used during the computation.

**Definition 10** ***(End relation)*** *An end relation* ***ER*** *is computed from a relevant join* ***RJ*** *by projecting out unnecessary columns: Let $CR$ be the central relation of the schema with primary key $PK$. The set of attributes $ATT$ is defined as the union of $PK$ and all attributes in the recipient's transcript. Then, the end relation $ER = \pi_{ATT}(RJ)$.*

Consistent with previous chapters, we model an adversary that has background knowledge regarding not only quasi-identifier values of individuals, but also links

| studentID | major | YoB | GPA | class | grade |
|---|---|---|---|---|---|
| S1 | Computer Science | 1994 | 3.26 | CS111 | B |
| S1 | Computer Science | 1994 | 3.26 | CS131 | A |
| S1 | Computer Science | 1994 | 3.26 | CS143 | A- |
| S2 | Computer Science | 1995 | 3.48 | CS111 | A- |
| S2 | Computer Science | 1995 | 3.48 | CS132 | A- |

Figure 5.1: An end relation for recipient $R_2$

between their tuples, e.g., how many classes each student took, how many evaluations they gave for each class etc.. We make the following observation that is critical to this adversarial model:

**Observation 5** *An adversary that knows how many tuples an individual has in a hierarchical database can use repeated sensitive values in the published dataset as identifying information, if sensitive attributes are not subject to generalizations and suppressions.*

This observation is overlooked by most of the previous work that acknowledges multiple tuples per individual may exist in a database, e.g., [8], [53], [46]. The primary reason that this is significant in our case is because our *end relation*s will contain sensitive attributes that have to be repeated in multiple tuples (See Fig. 5.1 for a sample end relation calculated for the data in Fig. 2.3), whereas existing literature often disregards such cases. This does not make the situation any less applicable to a real-life scenario: Consider the credit card transaction relation *(YoB, zipcode, occupation, salary, vendor, amount)*, where *salary* and *amount* are sensitive. Bob may have numerous credit card transactions from different vendors during a time period in which his salary stays the same. If this exact salary value is observed in $n$ tuples, all other salary values are observed in less than $n$ tuples and the adversary knows that Bob has made $n$ transactions, he successfully locates all of Bob's transactions.

| major | YoB | GPA | class | grade |
|---|---|---|---|---|
| Computer Science | 1990s | 3.26 | CS Undergrad | B |
| Computer Science | 1990s | 3.26 | CS Undergrad | A |
| Computer Science | 1990s | 3.26 | CS Undergrad | A- |
| Computer Science | 1990s | 3.48 | CS Undergrad | A- |
| Computer Science | 1990s | 3.48 | CS Undergrad | A- |

Figure 5.2: 5-anonymized table for the data in Fig. 5.1

**Observation 6** *Reoccurring sensitive values in a QI-wise properly k-anonymized end relation may cause privacy leaks.*

Observation 6 is a direct consequence of Observation 5. We present an example to demonstrate this point: Fig. 5.2 is 5-anonymized according to quasi-identifiers *major, YoB* and *class*. Its sensitive values are left intact. Join keys are not part of the output (recall our discussion regarding a recipient not needing "ownership") and are therefore removed. An adversary that knows Alice has taken three classes immediately learns that Alice's GPA is 3.26.

As shown here, anonymity with regard to quasi-identifiers is not sufficient, sensitive values also have to be anonymized. There are a number of ways in which this can be done: (1) assume that a generalization hierarchy is available for sensitive attributes, (2) use the "buckets" of distinct values in $l$-diversity as domain generalization hierarchies, or (3) a rather costly solution would be to suppress (replace with *) any set of sensitive values that violate indistinguishability. The third option might be useful in cases where generalizations are unacceptable, but it should be noted that a suppression would receive the highest penalty possible for the suppressed data cell.

We should point out that generalizations of sensitive attributes never convey more information to an adversary than leaving them intact. In other words, there are no "side effects" of generalizing GPA values, e.g., the GPA values in Fig. 5.2 can be generalized to "(3.00-4.00]". If they are left as is, though, anyone

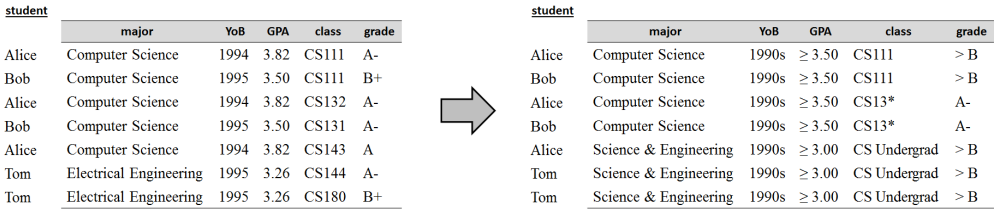
analyzing the database would still be able to trivially deduce that all observed values are between 3.00 and 4.00. If anything, generalizing sensitive values can hide information. How much information it can hide depends on the procedure used to anonymize them; as always, there is a trade-off between data utility and individuals' privacy here.

It is argued earlier in Chapter 2 and also in [37] that single table $k$-anonymity fails in cases where multiple tuples regarding the same person end up in the same equivalence class (An equivalence class is a group of tuples that are indistinguishable from one another). We therefore propose to apply a special case of (X,Y)-anonymity [46] to our problem.

**Definition 11** ***((X,Y)-anonymity)*** *Let $X$ and $Y$ be two disjoint sets of attributes (i.e., columns) in a relation $R$. Let $t$ be a tuple observed in $R$, and $s$ be $\pi_X(t)$, i.e., projection of $t$ on attributes in $X$. The "anonymity" of $s$ with respect to $Y$, denoted $\alpha_Y(s)$, is calculated as: $\alpha_Y(s) = |\pi_Y(\sigma_s(R))|$. Let the function $A_Y(X)$ be defined as $A_Y(X) = min\{\alpha_Y(s), \forall s \in X\}$. Then, we say that $R$ satisfies (X,Y)-anonymity for an integer parameter $k$, if $A_Y(X) \geq k$.*

Let $ATT$ be the set of all attributes in our end relation ER. To anonymize ER, we use (X,Y)-anonymity with $Y = PK$ and $X = ATT - PK$. $k$ is a variable privacy parameter (as in $k$-anonymity).

Consider the end relation built for recipient $R_2$ (e.g., Fig. 5.1), with transcript: *(studentID, major, YoB, GPA, class, grade)*. According to our definitions, we set $Y$={*studentID*} and $X$={*major, YoB, GPA, class, grade*}. Say that $s$ has the following value on $X$: *s=(Computer Science, 1990s, ≥3.50, CS131, A-)*, i.e., it is just a tuple of values on all attributes in X. $\alpha_Y(s)$ computes the distinct number of persons in the equivalence class of $s$. $A_Y(X)$ calculates the number of distinct persons in all equivalence classes in the relation, and picks the smallest value. The end relation is successfully (X,Y)-anonymized with parameter $k$, if $A_Y(X) \geq k$.



| student | major | YoB | GPA | class | grade |
|---|---|---|---|---|---|
| Alice | Computer Science | 1994 | 3.82 | CS111 | A- |
| Bob | Computer Science | 1995 | 3.50 | CS111 | B+ |
| Alice | Computer Science | 1994 | 3.82 | CS132 | A- |
| Bob | Computer Science | 1995 | 3.50 | CS131 | A- |
| Alice | Computer Science | 1994 | 3.82 | CS143 | A |
| Tom | Electrical Engineering | 1995 | 3.26 | CS144 | A- |
| Tom | Electrical Engineering | 1995 | 3.26 | CS180 | B+ |

| student | major | YoB | GPA | class | grade |
|---|---|---|---|---|---|
| Alice | Computer Science | 1990s | ≥ 3.50 | CS111 | > B |
| Bob | Computer Science | 1990s | ≥ 3.50 | CS111 | > B |
| Alice | Computer Science | 1990s | ≥ 3.50 | CS13* | A- |
| Bob | Computer Science | 1990s | ≥ 3.50 | CS13* | A- |
| Alice | Science & Engineering | 1990s | ≥ 3.00 | CS Undergrad | > B |
| Tom | Science & Engineering | 1990s | ≥ 3.00 | CS Undergrad | > B |
| Tom | Science & Engineering | 1990s | ≥ 3.00 | CS Undergrad | > B |

Figure 5.3: An input end relation and its (X,Y)-anonymization with $k = 2$

Similar to the privacy parameters $k$ in $k$-anonymity and $l$ in $l$-diversity, we can experiment with varying $k$ in (X,Y)-anonymity.

In Fig. 5.3 we present an example for (X,Y)-anonymity. We have Alice who took 3 classes, and Bob and Tom who took 2 classes. Alice, Bob and Tom each have unique primary keys in the database. (We use their names as primary keys in our example.) We build three equivalence classes using (X,Y)-anonymity, the first two containing two tuples and the last one containing three tuples. Notice that the last equivalence class contains 2 tuples from Tom, which implies that this equivalence class itself is also not 3-anonymous but 2-anonymous according to the (X,Y)-anonymity definition.

An interesting observation is that DGHs and bucketization are acceptable to generalize sensitive attributes, but natural generalization hierarchies and generalization lattices [36] are not appropriate. DGHs and buckets yield non-overlapping generalizations, which is a crucial point. For example, Tom (whose GPA is 3.26) has two tuples, and each could have been placed in different equivalence classes. If we allow overlapping generalizations, one equivalence class might generalize Tom's GPA to [3.00-4.00], and the other one [2.50-3.50]. An adversary, linking Tom to these two equivalence classes, would be able to compute the intersection of these two generalizations, which is [3.00-3.50]. This inference can be avoided by the use of non-overlapping generalizations, e.g., Fig. 5.4.

It can be claimed that it is possible to learn Alice and Bob's grade by looking at the second equivalence class, and this is a privacy leak. However, this attack

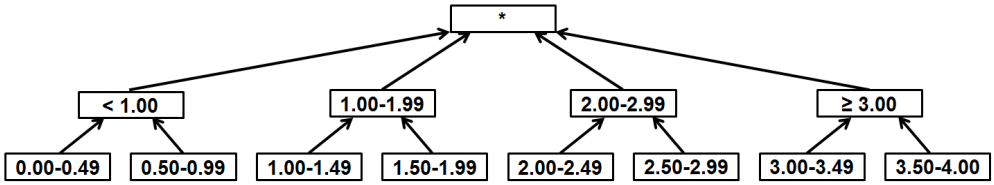



Figure 5.4: Domain generalization hierarchy for GPA

is possible simply because of the lack of diversity in sensitive values. The notion of *anonymity*, as discussed earlier, disregards the distribution of sensitive values. One needs to extend our work to support *diversity* to thwart such threats.

## 5.3 Overview

Before proceeding further, let us briefly summarize the discussion so far. We are attacking the problem of privacy-preserving publishing of certain columns and relations in a hierarchically organized (i.e., snowflake schema) multi-relational database. We assume that recipients of our data have no desire to learn about the hierchical relationship between tuples (otherwise our $k$-anonymity solution in Chapter 3 would suffice).

Our methodology to solve this problem can be studied in several steps: (1) Flatten the hierarchical dataset by computing a *relevant join*. (2) Find the *end relation* of the relevant join. The end relation will contain all attributes that are sent to the data recipient, plus the primary key attribute of the dataset. (3) (X,Y)-anonymize the end relation using Y = primary key of the central relation, and X = all attributes in the end relation minus the primary key. (4) Remove the primary key column (i.e., project it out) from the (X,Y)-anonymized end relation and publish the result to the data recipient.

The rest of this chapter will focus on how (X,Y)-anonymity can be achieved. We assume that an end relation ready to be (X,Y)-anonymized has been computed, and provide an algorithm that (X,Y)-anonymizes its input.

## 5.4 Achieving Anonymity

A **tuple** is a data tuple from the end relation, of the format: *(PKid,* $QI_1$*,* $QI_2$*, ..,* $QI_n$*,* $S_1$*,* $S_2$*, .. *$S_n$*)*. *PKid* is an individual's primary key identifier, e.g., Alice has id=1, Bob has id=2 etc.. Both $QI_i$s (quasi-identifier values) and $S_j$s (sensitive values) are subject to generalizations. Let $tuple[i]$ denote the value of a tuple's i'th column/attribute, and $tuple.ID$ denote the PKid value of the tuple.

We define the function **anonymize** as follows:

**anonymize**($t_1$,$t_2$): Takes tuples $t_1$ and $t_2$ as inputs. Modifies them in a way such that resulting $t_1$ and $t_2$ will be indistinguishable in terms of QI and S attributes, with lowest information loss possible. Additionally, assume that it has a return value which is a single tuple containing the end result (i.e., generalized version of $t_1$ and $t_2$).

Since every QI and S has its own generalization hierarchy/lattice that is independent of other attributes, **anonymize** generalizes each $t[i]$ (for all $i$ that is not PKid) one by one. Achieving minimum cost in each column (according to a pre-defined cost metric, e.g., LM) ensures minimum cost between a pair of tuples.

Similar to our previous solutions in Chapters 2 and 3, let **copy**($t_i$) perform a deep copy of tuple $t_i$ and return this copy. Also, assume that **cost**($t_i$) calculates the information loss in $t_i$ .

We designed a clustering algorithm to solve the problem, but before moving on to the algorithm itself, we should define some notation. Let $c_i$ denote the i'th cluster. A cluster $c_i$ has:

- A cluster representative, denoted $c_i.rep$. This is a single tuple that summarizes the whole cluster: What would the outcome be if all tuples within $c_i$ were **anonymize**d? In other words, it holds the current "state" of the tuples within $c_i$.

- Set of tuples within the cluster, denoted $c_i.tuples$. These tuples can be kept with or without updates (i.e., the same as they were in the input or anonymized as new tuples are added to cluster). Updates are not necessary on-the-fly since the cluster representative already keeps the current state of the cluster. (We choose not to update them while the clustering procedure is running in order to obtain better performance.)

- Set of unique PKids, denoted $c_i.pk$. This set contains the primary key IDs of individuals whose tuples are in $c_i$.

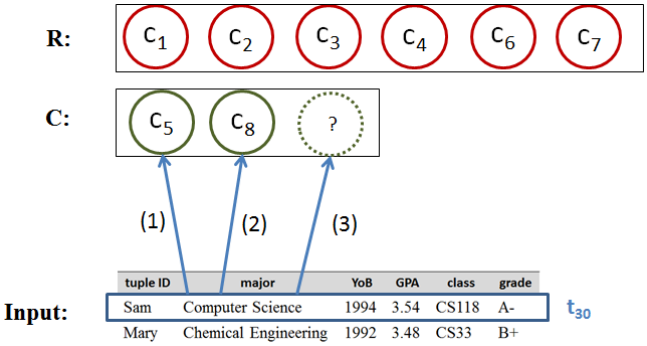
Our clustering algorithm is presented formally in Algorithm 5. We perform a single pass over all tuples in the input. For each tuple, we find the most suitable cluster in the list of "active" clusters, i.e., clusters that were previously initialized and are not yet full. Suitability is determined by measuring the potential information loss (i.e., data utility cost) of placing that tuple in a certain cluster. At this stage, we try to have as many distinct individuals in a cluster as possible to satisfy (X,Y)-anonymity, hence the primary key enforcement on line 6. A sensitivity parameter $\gamma$ is employed by the algorithm: If there are no active clusters to place this tuple in, or if the cluster that is "nearest" to this tuple is further than $\gamma$ away, then a new cluster is initialized using the tuple. The newly initialized cluster is immediately added to the list of active clusters. On the other hand, if a suitable cluster could be found (i.e., distance between the closest cluster and the tuple was less than $\gamma$) the tuple is added to this suitable cluster. After this operation, we check whether the updated cluster satisfies (X,Y)-anonymity for privacy parameter $k$, and if so, we mark this cluster as full and retire it.

After all tuples are processed, we are left with clusters that have less than $k$ elements and therefore do not satisfy (X,Y)-anonymity. We pick a cluster $c_i$ from the list of remaining clusters and find the most suitable $c_j$ to match it with, again by calculating the potential information loss that this matching would incur. The

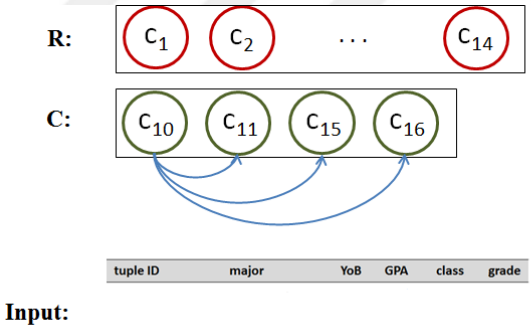
**Algorithm 5** Clustering for (X,Y)-anonymity

**Input:** A set of tuples $t_1$, .., $t_n$, privacy parameter $k$ and sensitivity parameter $\gamma$

```
1:  procedure XYCLUSTERING
2:      Initialize C ← {}, list of active clusters
3:      Initialize R ← {}, list of clusters that are full
4:      for tuple t_i in input do
5:          Find c_i ∈ C such that dist = cost(anonymize(copy(t_i), copy(c_i.rep)))
6:                is minimum and t_i.ID ∉ c_i.pk
7:          if |C| = 0 or dist ≥ γ then
8:              Create new cluster c_k, where c_k.rep = t_i and c_k.tuples ← {t_i}
9:              Add t_i.ID to c_k.pk
10:             C ← C + c_k
11:         else
12:             anonymize(t_i, c_i.rep)
13:             c_i.pk ← c_i.pk ∪ t_i.ID
14:             c_i.tuples ← c_i.tuples ∪ t_i
15:             if |c_i.pk| ≥ k then
16:                 C ← C − c_i
17:                 R ← R + c_i
18:     while |C| ≥ 2 do
19:         Pick c_i ∈ C and find c_j ∈ C such that cost(anonymize(copy(c_i.rep),
20:               copy(c_j.rep))) is minimum
21:         anonymize(c_i.rep, c_j.rep)
22:         c_i.tuples ← c_i.tuples ∪ c_j.tuples
23:         c_i.pk ← c_i.pk ∪ c_j.pk
24:         C ← C − c_j
25:         if |c_i.pk| ≥ k then
26:             C ← C − c_i
27:             R ← R + c_i
28:     if |C| = 1 then
29:         Let c_i be the only cluster left in C
30:         Suppress tuples in c_i
31:         R ← R + c_i
32:     return R
```




(a) Consuming unclustered tuples from the input (lines 4-17). At this point, $t_{30}$ arrives and potential next steps are labeled (1)-(3). Step (1) is executed if $dist(t_{30}, c_5.rep) < dist(t_{30}, c_8.rep)$ and $dist(t_{30}, c_5.rep) < \gamma$. Step (2) is executed if $dist(t_{30}, c_8.rep) < dist(t_{30}, c_5.rep)$ and $dist(t_{30}, c_8.rep) < \gamma$. Step (3) is executed, i.e., a new cluster is initialized, if $\gamma > dist(t_{30}, c_5.rep)$ and $\gamma > dist(t_{30}, c_8.rep)$. If (1) or (2) is executed and the resulting cluster becomes full (i.e., size $\geq k$), it is removed from $C$ and added to $R$.




(b) Merging left-over clusters in $C$ after the input is exhausted (lines 18-31). A cluster is chosen (in this case, $c_{10}$) and its distance to other clusters in $C$ are computed. The cluster with minimum distance is then merged with $c_{10}$.

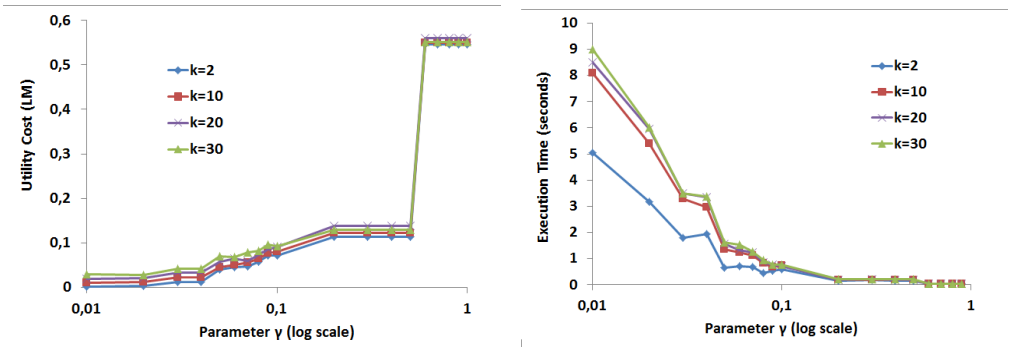
Figure 5.5: Sample execution for Algorithm 5.

best candidate $c_j$ is matched with $c_i$, and they are merged. If the resulting cluster satisfies (X,Y)-anonymity, then it can be deemed full and removed from the list of active clusters. This process continues until there are less than 2 clusters left. At that point, there might be one cluster remaining (or there might be zero). All tuples in this left-over cluster are suppressed.

We use $\gamma$ as a parameter to control the behavior of the algorithm. Depending on the utility cost metric used, $\gamma$ can take values between 0 and the maximum penalty a tuple can receive. (E.g., in LM, tuples' costs are anonymized to a value between 0 and 1.) $\gamma$ determines how likely tuples are to be clustered together during the initial pass. With $\gamma = 0$, all tuples will cause a new cluster to be initialized, and therefore the cluster merging process will start only when the algorithm reaches line 18. This requires higher memory (since all tuples have to be kept in memory and no cluster will be able to "retire" until line 18) and is likely to cause longer execution time. On the other hand, higher values of $\gamma$ will cause very distant tuples to merge together, leading to higher utility loss. In the most extreme case, when $\gamma = 1$ (or whatever the highest penalty is), there will be at most 1 active cluster during the algorithm at all times. Every tuple that is read from the input will have to be clustered with whatever the current active cluster is, no matter how inconvenient the result may be. This would yield good performance, but is probably way too insensitive to obtain reasonable utility cost.

## 5.5 Experiments

We re-use our datasets from the previous chapters. We model two recipients based on our data: $R_1$(age, GPA, course, grade) which needs data from the first two relations and $R_2$(age, GPA, course, grade, book, price) which needs data from all three relations.




(a) Utility cost vs $\gamma$ (b) Execution time vs $\gamma$

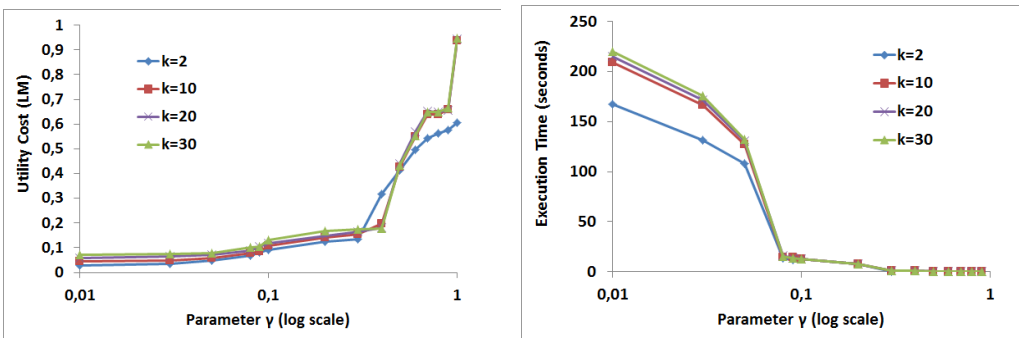
Figure 5.6: Results for $R_1$ with $\gamma$ ranging from 0 to 1

### 5.5.1 Effect of $\gamma$

We run our experiments with various values for the $\gamma$ parameter on the large dataset in order to obtain robust results regarding accuracy and speed.

In Fig. 5.6, utility costs and execution times are graphed against different values of $\gamma$. These values are for recipient $R_1$. We graph seperate curves for each $k$: $k = 2$, $k = 10$, $k = 20$ and $k = 30$. We observe that all curves have similar behavior. The x-axis is drawn in logarithmic scale to better focus on small values of $\gamma$. As expected, smaller $\gamma$ have higher execution time, but less utility cost. Utility cost increases steadily between $\gamma = 0.01$ to $\gamma = 0.1$, and then relatively more rapidly from $\gamma = 0.1$ to $\gamma = 0.5$. Afterwards, there is a steep increase, probably because of the fact that the algorithm becomes way too insensitive to utility loss of tuples and tries to add almost every tuple in its input to an existing cluster during the initial pass (i.e., lines 4-17). Seeing that all experiments could finish in under less than a minute, it is reasonable to use smaller values of $\gamma$ for recipient $R_1$.

The algorithm is not as fast for $R_2$, but it can still complete in just a few minutes. Since $R_2$ has an extra relation that $R_1$ does not (the relation that contains books and prices), both the number of tuples and the number of QIs




(a) Utility cost vs $\gamma$ (b) Execution time vs $\gamma$

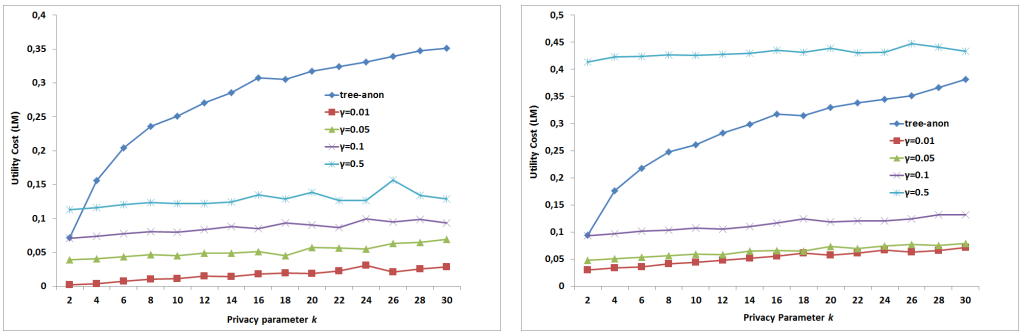
Figure 5.7: Results for $R_2$ with $\gamma$ ranging from 0 to 1

to be generalized are larger in $R_2$'s case. As a result, it takes several minutes to anonymize the dataset for $R_2$. We noticed that one of the main reasons causing worse performance (i.e., bottleneck) was the average number of active clusters during the initial pass. In order to find a suitable cluster for an incoming tuple, the algorithm (on line 5) checks all of the active clusters at that point in time. Clearly, a large number of clusters implies more computation for every tuple in the input. While running our experiments, we saw that over 3000 clusters were active (on average) when $\gamma = 0.01$, whereas with $\gamma = 0.1$ the number of active clusters was reduced to 500 (on average). Still, we believe that the algorithms are efficient, given that they can handle several thousand tuples in under 4 minutes for all $\gamma$. All results regarding efficiency and accuracy for $R_2$ are provided in Fig. 5.7.

We see that the utility costs in $R_2$'s case are similar to the trends observed in $R_1$'s graphs. In both cases, there is a clear trade-off between execution time and utility cost, which can be controlled by the $\gamma$ parameter. Costs are usually below or around 0.1 for $\gamma \leq 0.1$.

### 5.5.2 Comparison with Anonymity of Trees

In this section, we compare our $k$-anonymity algorithm for trees (that was presented in Chapter 2 and we will refer to it in this section as *tree-anonymity*) and



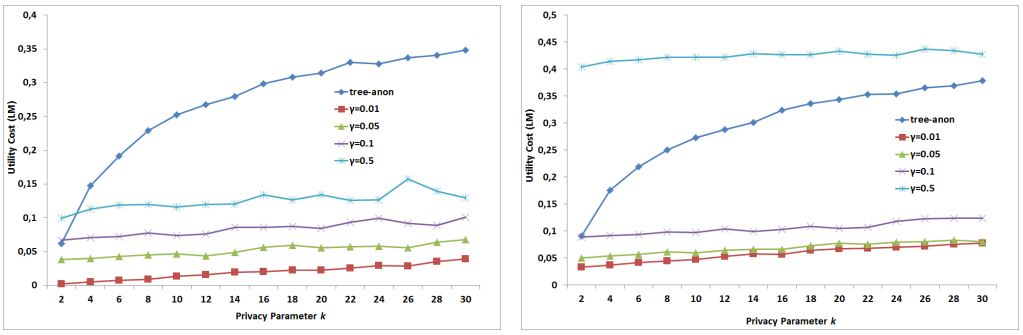

(a) Recipient $R_1$ (b) Recipient $R_2$

Figure 5.8: Utility costs of anonymization algorithms using the large dataset

our (X,Y)-anonymity algorithm for partial publication. Let us re-iterate that the two problems are essentially different, and the problem in this chapter can be seen as a "relaxed" version of the problem we considered in Chapter 2. But, we have also argued that tree-anonymity is an applicable solution to the problem in this chapter, yet it is a solution that is unnecessarily demanding. We now try to evaluate this claim using our algorithms.

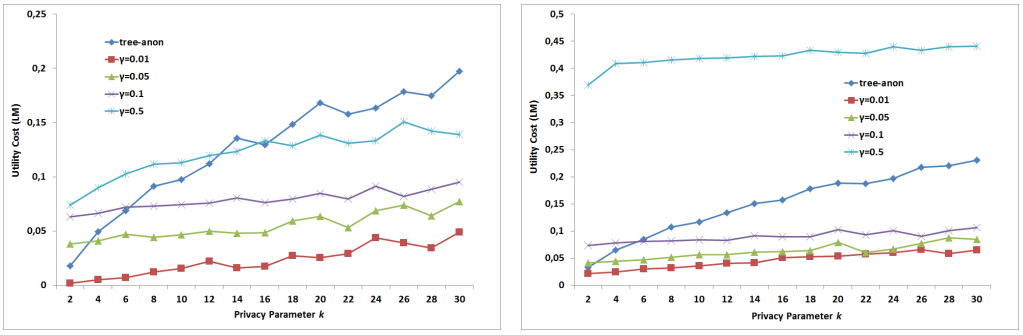
For a fair comparison, we calculate the LM costs for $R_1$ and $R_2$ seperately, taking into account only the attributes and fields that are published to the recipient in question. For example, in the case of $R_1$, we compute the cost of the tree-anonymized dataset using only the first two levels and disregard the costs of the third level. Therefore the LM cost of the tree-anonymized dataset is different for $R_1$ and $R_2$. As observed in Fig. 5.8, Fig. 5.9 and Fig. 5.10, the costs for $R_1$ are usually slightly lower than the costs for $R_2$.

Our comparison shows that with smaller values of $\gamma$, we can often obtain much better anonymizations using (X,Y)-anonymity than tree-anonymity. The discrepancy between tree-anonymity and (X,Y)-anonymity becomes significant especially when $k$ is large. Even with $\gamma = 0.1$, which was previously shown to be very efficient considering the size of the datasets, (X,Y)-anonymity outperforms tree-anonymity by factors of 3-4. Utility costs decrease further for $\gamma < 0.1$, although runtimes




(a) Recipient $R_1$ (b) Recipient $R_2$

Figure 5.9: Utility costs of anonymization algorithms using the medium dataset




(a) Recipient $R_1$ (b) Recipient $R_2$

Figure 5.10: Utility costs of anonymization algorithms using the small dataset

increase (almost exponentially). On the other hand, a relatively large $\gamma$ is undesirable, e.g., $\gamma = 0.5$ causes utility costs to be higher than tree-anonymity for $R_2$. These experiments also strengthen the idea that not only the value of privacy parameter $k$ is important when anonymizing a dataset (i.e., as we move from left to right on each curve, utility costs tend to increase), but also parameter $\gamma$ dictates the accuracy and effectiveness of our algorithm.

Given the circumstances, a user might wonder what the most appropriate $\gamma$ value for a dataset is. We believe that this is highly dependent on the computation power and time available to the user. In short, our experiments suggest that lower values of $\gamma$ almost always cause better anonymizations, but higher execution time. We should remind the reader that these experiments were run on commodity hardware. In any case, we obtain much better execution times with (X,Y)-anonymity compared to tree-anonymity. On the large dataset, tree-anonymity took around 15 minutes to run, whereas here, even with small $\gamma$, it takes no more than 4 minutes to anonymize the whole data.

We have validated that we can come up with algorithms that are both faster and more accurate for the more relaxed problem of (X,Y)-anonymity, compared to tree-anonymity. Although this does not formally prove our point about partial publication being an "easier" problem than tree-anonymity (one can very well come up with an algorithm for tree-anonymity that performs better than ours), we believe that the obvious discrepancy between the results we obtain for the two cases is a strong indication that supports our claim.

# CHAPTER 6

# Conclusion and Future Work

This thesis investigated the problem of privacy-preserving publishing of hierarchically structured data. We formalized and applied well-known definitions of privacy on hierarchical data in two distinct cases: (1) where the structure of the data must be explicitly preserved, and (2) where the structure is irrelevant and the data recipient is interested only in certain relations in the database. In both cases we assumed adversaries had full knowledge regarding individuals' quasi identifiers and potential links between their data records in different relations (e.g., how many books a student bought for a certain class). We designed and implemented generic algorithms that are independent of the underlying database and utility cost metric. We experimented on practical synthetic databases that were generated using an actual real-life sample obtained from a university.

A particular highlight is that even though we used the LM metric to evaluate utility costs, our approach is suitable for arbitrary monotonic cost metrics. One can use, for example, a cost metric that penalizes certain levels or relations more than others. (E.g., generalizations in the root node may have higher cost than the leaves.) It is also possible to create reasonable heuristics and extensions, in addition to what is presented in this thesis. In particular, domain-specific heuristics (e.g., applicable to education data or health data only) might be useful in real world anonymizations.

We believe that there are many exciting possibilities for future work. Some of these issues would be: (1) Speeding up execution of our algorithms using more ef-

ficient data structures. Our solutions are slower for large datasets (e.g., ones that contain $> 15000$ tuples) which is an issue we plan to address. Potential remedies might include parallel execution and/or application of heuristics. (2) Utilizing other applicable definitions of privacy, such as $t$-closeness or probabilistic definitions of $l$-diversity. Especially, differential privacy [15][9] has recently gained quite some attention from the research community. (3) We allowed no data perturbation, i.e., no noise could be added to the database. Data perturbation can be a useful tool in certain situations, and may (or may not) provide end results with higher data utility.

Finally, we also wish to consider continuous and incremental data publishing in future work. XML streams and XML processing engines [4] are popular sources of growing and accumulating data, which may need to be anonymized before being shared with a third party. [8] and [53] already examine the issue of privacy in streaming data, where data arrives in a continuous stream. In [39] and [7], databases grow incrementally, and each incremental update needs to be published taking into account what has been previously published. These solutions assume single-relational data, but it would be interesting to apply their ideas to hierarchical data.

# References


[1] Aggarwal, G., Feder, T., Kenthapadi, K., Khuller, S., Panigrahy, R., Thomas, D., & Zhu, A. (2006, June). Achieving anonymity via clustering. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (pp. 153-162). ACM.

[2] Aggarwal, G., Feder, T., Kenthapadi, K., Motwani, R., Panigrahy, R., Thomas, D., & Zhu, A. (2005). Approximation algorithms for k-anonymity. *Journal of Privacy Technology (JOPT)*.

[3] Aggarwal, G., Panigrahy, R., Feder, T., Thomas, D., Kenthapadi, K., Khuller, S., & Zhu, A. (2010). Achieving anonymity via clustering. *ACM Transactions on Algorithms (TALG)*, 6(3), 49.

[4] Avila-Campillo, I., Green, T. J., Gupta, A., Onizuka, M., Raven, D., & Suciu, D. (2002). XMLTK: An XML toolkit for scalable XML stream processing.

[5] Bayardo, R. J., & Agrawal, R. (2005, April). Data privacy through optimal k-anonymization. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)* (pp. 217-228). IEEE.

[6] Brand, R. (2002). Microdata protection through noise addition. In *Inference control in statistical databases* (pp. 97-116). Springer Berlin Heidelberg.

[7] Byun, J. W., Sohn, Y., Bertino, E., & Li, N. (2006). Secure anonymization for incremental datasets. In *Secure Data Management* (pp. 48-63). Springer Berlin Heidelberg.

[8] Cao, J., Carminati, B., Ferrari, E., & Tan, K. L. (2011). Castle: Continuously anonymizing data streams. *IEEE Transactions on Dependable and Secure Computing*, 8(3), 337-352.

[9] Chen, R., Mohammed, N., Fung, B. C., Desai, B. C., & Xiong, L. (2011). Publishing set-valued data via differential privacy. *Proceedings of the VLDB Endowment*, 4(11), 1087-1098.

[10] Cicek, A. E., Nergiz, M. E., & Saygin, Y. (2014). Ensuring location diversity in privacy-preserving spatio-temporal data publishing. *The VLDB Journal*, 23(4), 609-625.

[11] Cox, L. H. (1980). Suppression methodology and statistical disclosure control. *Journal of the American Statistical Association*, 75(370), 377-385.

[12] Dalenius, T. (1986). Finding a needle in a haystack-or identifying anonymous census record. *Journal of official statistics*, 2(3), 329-336.

[13] Domingo-Ferrer, J., & Torra, V. (2005). Ordinal, continuous and heterogeneous k-anonymity through microaggregation. *Data Mining and Knowledge Discovery, 11(2)*, 195-212.

[14] Domingos, P. (2003). Prospects and challenges for multi-relational data mining. *ACM SIGKDD explorations newsletter*, 5(1), 80-83.

[15] Dwork, C. (2008). Differential privacy: A survey of results. In *Theory and Applications of Models of Computation* (pp. 1-19). Springer Berlin Heidelberg.

[16] Dzeroski, S. (2003). Multi-relational data mining: an introduction. *ACM SIGKDD Explorations Newsletter, 5(1)*, 1-16.

[17] Fung, B., Wang, K., Chen, R., & Yu, P. S. (2010). Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (CSUR), 42(4)*, 14.

[18] Ganesan, P., Garcia-Molina, H., & Widom, J. (2003). Exploiting hierarchical domain structure to compute similarity. *ACM Transactions on Information Systems (TOIS), 21(1)*, 64-93.

[19] Ghinita, G., Karras, P., Kalnis, P., & Mamoulis, N. (2007, September). Fast data anonymization with low information loss. In *Proceedings of the 33rd International Conference on Very Large Data Bases* (pp. 758-769). VLDB Endowment.

[20] Han, J., & Kamber, M. (2006). *Data Mining, Southeast Asia Edition: Concepts and Techniques*. Morgan Kaufmann.

[21] He, Y., & Naughton, J. F. (2009). Anonymization of set-valued data via top-down, local generalization. *Proceedings of the VLDB Endowment*, 2(1), 934-945.

[22] Iyengar, V. S. (2002, July). Transforming data to satisfy privacy constraints. In *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 279-288). ACM.

[23] Kanza, Y., Mendelzon, A. O., Miller, R. J., & Zhang, Z. (2006). Authorization-transparent access control for XML under the non-Truman model. In *Advances in Database Technology-EDBT 2006* (pp. 222-239). Springer Berlin Heidelberg.

[24] Landberg, A. H., Nguyen, K., Pardede, E., & Rahayu, J. W. (2014). $\delta$-Dependency for privacy-preserving XML data publishing. *Journal of biomedical informatics*.

[25] LeFevre, K., DeWitt, D. J., & Ramakrishnan, R. (2005, June). Incognito: Efficient full-domain k-anonymity. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (pp. 49-60). ACM.

[26] LeFevre, K., DeWitt, D. J., & Ramakrishnan, R. (2006). Mondrian multidimensional k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE '06)*. IEEE.

[27] Li, J., Tao, Y., & Xiao, X. (2008, June). Preservation of proximity privacy in publishing numerical sensitive data. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (pp. 473-486). ACM.

[28] Li, N., Li, T., & Venkatasubramanian, S. (2007, April). t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In *ICDE '07* (Vol. 7, pp. 106-115).

[29] Liu, J., & Wang, K. (2010, March). On optimal anonymization for l+-diversity. In *26th IEEE International Conference on Data Engineering (ICDE 2010)*, (pp. 213-224). IEEE.

[30] Liu, K., & Terzi, E. (2008, June). Towards identity anonymization on graphs. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (pp. 93-106). ACM.

[31] Machanavajjhala, A., Kifer, D., Gehrke, J., & Venkitasubramaniam, M. (2007). l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD), 1(1)*, 3.

[32] Meyerson, A., & Williams, R. (2004, June). On the complexity of optimal k-anonymity. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '04)* (pp. 223-228). ACM.

[33] Narayanan, A., & Shmatikov, V. (2008, May). Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, 2008. (pp. 111-125). IEEE.

[34] Nergiz, M. E., Atzori, M., & Saygin, Y. (2008, November). Towards trajectory anonymization: a generalization-based approach. In *Proceedings of the SIGSPATIAL ACM GIS 2008 International Workshop on Security and Privacy in GIS and LBS* (pp. 52-61). ACM.

[35] Nergiz, M. E., Atzori, M., & Clifton, C. (2007, June). Hiding the presence of individuals from shared databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data* (pp. 665-676). ACM.

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

[37] Nergiz, M. E., Clifton, C., & Nergiz, A. E. (2009). Multirelational k-anonymity. *IEEE Transactions on Knowledge and Data Engineering*, 21(8), 1104-1117.

[38] Park, H., & Shim, K. (2007, June). Approximate algorithms for k-anonymity. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data* (pp. 67-78). ACM.

[39] Pei, J., Xu, J., Wang, Z., Wang, W., & Wang, K. (2007, July). Maintaining k-anonymity against incremental updates. In *19th International Conference on Scientific and Statistical Database Management (SSDBM)*, 2007. IEEE.

[40] Reiss, S. P. (1984). Practical data-swapping: The first steps. *ACM Transactions on Database Systems (TODS)*, 9(1), 20-37.

[41] Samarati, P., & Sweeney, L. (1998). *Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression.* Technical report, SRI International.

[42] Sweeney, L. (2000). *Uniqueness of simple demographics in the US population.* Technical report, Carnegie Mellon University.

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

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

[45] Wang, K., Fung, B. C., & Philip, S. Y. (2007). Handicapping attacker's confidence: an alternative to k-anonymization. *Knowledge and Information Systems*, 11(3), 345-368.

[46] Wang, K., & Fung, B. (2006, August). Anonymizing sequential releases. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 414-423). ACM.

[47] Wong, R. C. W., Li, J., Fu, A. W. C., & Wang, K. (2006, August). ($\alpha$, k)-anonymity: an enhanced k-anonymity model for privacy preserving data publishing. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 754-759). ACM.

[48] Terrovitis, M., Mamoulis, N., & Kalnis, P. (2008). Privacy-preserving anonymization of set-valued data. *Proceedings of the VLDB Endowment*, 1(1), 115-125.

[49] Xiao, X., & Tao, Y. (2006, September). Anatomy: Simple and effective privacy preservation. In *Proceedings of the 32nd International Conference on Very Large Data Bases (pp. 139-150)*. VLDB Endowment.

[50] Xiao, X., Yi, K., & Tao, Y. (2010, March). The hardness and approximation algorithms for l-diversity. In *Proceedings of the 13th International Conference on Extending Database Technology* (pp. 135-146). ACM.

[51] Xiong, H., Steinbach, M., & Kumar, V. (2006). Privacy leakage in multi-relational databases: a semi-supervised learning perspective. *The VLDB Journal - The International Journal on Very Large Data Bases*, 15(4), 388-402.

[52] Zheleva, E., & Getoor, L. (2008). Preserving the privacy of sensitive relationships in graph data. In *Privacy, security, and trust in KDD* (pp. 153-171). Springer Berlin Heidelberg.

[53] Zhou, B., Han, Y., Pei, J., Jiang, B., Tao, Y., & Jia, Y. (2009, March). Continuous privacy preserving publishing of data streams. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology* (pp. 648-659). ACM.

[54] Zhou, B., Pei, J., & Luk, W. (2008). A brief survey on anonymization techniques for privacy preserving publishing of social network data. *ACM SIGKDD Explorations Newsletter*, 10(2), 12-22.