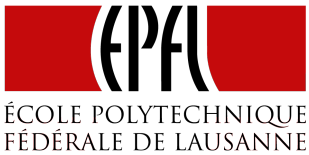
Privacy-Preserving Schema Reuse



Nguyen Quoc Viet Hung
Do Son Thanh
Nguyen Thanh Tam
Prof. Karl Aberer
School of Computer and Communication Science
EPFL

Technical Report
Distributed Information Systems Laboratory
07 June, 2013

Abstract

As the number of schema repositories grows rapidly and several web-based platforms exist to support publishing schemas, *schema reuse* becomes a new trend. Schema reuse is a methodology that allows users to create new schemas by copying and adapting existing ones. This methodology supports to reduce not only the effort of designing new schemas but also the heterogeneity between them. One of the biggest barriers of schema reuse is privacy concerns that discourage the participants from contributing their schemas. Addressing this problem, we develop a framework that enables privacy-preserving schema reuse. To this end, our framework supports users to define their own protection policies in the form of *privacy constraints*. Instead of showing original schemas, the framework returns an *anonymized schema* with maximal *utility* while satisfying these privacy constraints. To validate our approach, we empirically show the efficiency of different heuristics, the correctness of the proposed utility function, the computation time, as well as the trade-off between utility and privacy.

Contents

1

Introduction

Schema reuse is a new trend in creating schemas by allowing users to copy and adapt existing ones. The key driving forces behind schema reuse are the slight differences between schemas in the same domain; thus making the reuse becomes realistic. Reusing existing schemas supports to reduce not only the effort of creating a new schema but also the heterogeneity between schemas. Moreover, as the number of publicly available schema repositories (e.g. schema.org[1], Factual[2], Socrata[3], niem.gov[4]) grows rapidly and several web-based platforms (e.g. Freebase [5], Google Fusion Tables [6]) exists to support publishing schemas, reusing them becomes a great interest in both academic and industrial world.

One of the biggest barriers of reuse is the privacy concerns that discourage the participants from contributing their schemas [7]. In traditional approaches, all original schemas and their own attributes are presented to users [8]. However, in practical scenarios, the linking of attributes to their containing schemas, namely *attribute provenance*, is dangerous because of two reasons. First, providing the whole schema leads to privacy risks and potential attacks (e.g. SQL injection, see Appendix) on the owner database. Second, since some parts of a schema are the source of revenue and business strategy, the schema contributors want to protect these sensitive attributes to maintain their competitiveness. As a result, there is a need of developing new techniques to cope with these requirements.

In this paper, we develop a framework that protects attribute provenance in schema reuse. To this end, our framework enables users to define their own protection policies in terms of privacy constraints. Unlike previous works [8, 9, 10], we do not focus on finding and ranking schemas relevant to a user search query. Instead, our framework takes as input these relevant schemas and visualizes them in a unified view, namely *anonymized schema*, which satisfies pre-defined privacy constraints. Constructing such an anonymized schema is challenging because of three reasons. First, defining the representation for an anonymized schema is non-trivial. The anonymized schema should be concise enough to avoid overwhelming but also generic enough to provide comprehensive understanding. Second, for the purpose of comparing different anonymized

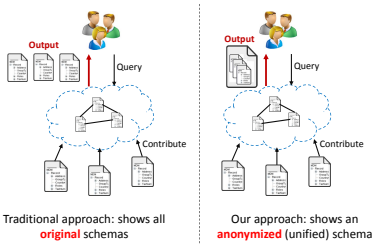


Figure 1.1: System Overview

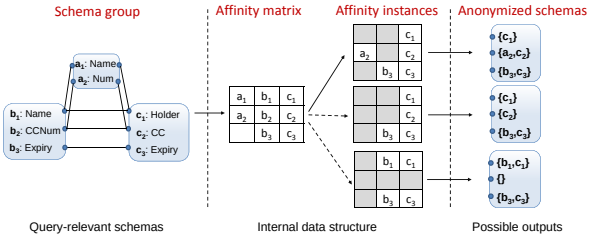


Figure 1.2: Solution Overview

schemas, we need to define a utility function to measure the amount of information they carry. The utility value must reflect the conciseness and the completeness of an anonymized schema. Third, finding an anonymized schema that maximizes the utility function and satisfies privacy constraints is NP-complete.

The key contributions of this paper are as follows. First, we model the problem of schema reuse with privacy constraints. In doing so, we propose a novel approach that constructs an anonymized schema (unified view) of original ones. In this approach, we introduce the concept of *affinity matrix* (represents a group of relevant schemas) and *presence constraint* (is a privacy constraint that translates human-understandable policies into mathematical standards). Then we show how to check whether an anonymized schema satisfies a pre-defined presence constraint using the probability theory.

Our second contribution is the development of a quantitative metric for assessing the quality of an anonymized schema. This metric captures two important aspects: (i) *attribute importance*—which reflects the popularity of an attribute—and (ii) *completeness*—which reflects the diversity of attributes in the anonymized schema. Both of them are combined in a single comprehensive notion, called *utility* of an anonymized schema.

As our final contribution, we show the intractability result for the problem of finding an anonymized schema with maximal utility, given a set of privacy constraints. Hence, we sketch out a heuristic-based algorithm and then improve its performance by caching redundant computations and decomposing the set of privacy constraints into inter-independent subsets, namely conflicting zones. Through experiments on real and synthetic data, we show the effectiveness of our algorithm with proposed heuristics.

The paper is organized as follows. Section 2 gives an overview of our approach. Section 3 formally introduces the notion of schema group, anonymized schema and privacy constraint. Section 4 shows how to measure the quality of an anonymized schema. Section 5 demonstrates the intractability result and the heuristic-based algorithm to the *maximizing anonymized schema* problem. Section 6 provides experiment results establishing the efficacy of our framework. Section 7 and 8 present related work and the conclusions.

2

Overview

Figure 1.1 provides an illustration of schema reuse systems. In these systems, there is a repository of schemas—which are contributed by many participants. These schemas have similar meanings but different structures and vocabularies. We focus on the scenario in which end-users want to design a new schema. To find relevant schemas for their design, they will explore the repository through search queries [8, 9, 10]. In traditional approaches, relevant schemas and their whole attributes are shown to users. However, revealing sensitive attributes and their linking to containing schemas could lead to privacy threats and unprivileged accesses (e.g. SQL injection, see Appendix). For this reason, we propose a novel approach that shows a unified view of original schemas, namely *anonymized schema*. In our approach, schema owners are given the rights to protect sensitive attributes by defining privacy constraints. The resulting anonymized schema has to be representative to cover original ones but opaque enough to prevent leaking the provenance of sensitive attributes (i.e. the linking to containing schemas).

Towards this approach, we propose a framework that enables privacy-preserving schema reuse. The input of our framework is a group of schemas—which is relevant to user query—and a set of privacy constraints—which translates human-understandable policies into mathematically sound standards. Respecting these constraints, the framework returns an anonymized schema with maximal utility, which reflects the amount of information it carries. The problem of constructing such an anonymized schema

Table 2.1: Summary of Important Notations

Symbol	Description
$\mathcal{S} = \{s_i\}$	a group of schemas
A_s	a set of attributes of the schema s
C	a set of attribute correspondences
M, I	affinity matrix, affinity instance
$\hat{S}$, $\hat{A}$	anonymized schema, abstract attribute
$u(\hat{S})$	utility of an anonymized schema
$\Gamma = \{\gamma_i\}$	a set of privacy constraints

is challenging and its details will be described in Section 5. Figure 1.2 depicts the simplified process of our solution for this problem, starting with a schema group and generated correspondences (solid lines) that indicates the semantic similarity between attributes. First, we represent a schema group by an internal data structure, called *affinity matrix*. In an affinity matrix, attributes in the same column belongs to the same schema, while attributes in the same row have similar meanings. Next, we derive various *affinity instances* by eliminating some attributes from the affinity matrix. For each affinity instance, we construct an anonymized schema with many *abstract attributes*. Each abstract attribute is a set of original attributes in the same row. Among constructed anonymized schemas, we select the one that maximizes the utility function and then present this "best" anonymized schema to user as final output. Table 2.1 summarizes important notations, which will described in the next section.

3

Model

In this section, we describe three important elements of our schema reuse approach: (i) schema group—a set of schemas relevant to a user search query, (ii) anonymized schema—a unified view of all relevant schemas in the group, and (iii) privacy constraint—a mean to represent human-understandable policies by mathematical standards. All these elements are fundamental primitives for potential applications built on top of our framework.

3.1 Schema Group

First of all, we model a schema as a finite set of attributes $A_s = \{a_1, ..., a_n\}$. Let $\mathcal{S} = \{s_1, ..., s_n\}$ be a schema group that is a set of schemas relevant to a specific search query. Let $A_\mathcal{S}$ denote the set of attributes in $\mathcal{S}$, i.e. $A_\mathcal{S} = \bigcup_i A_{s_i}$. Two schemas can share a common attribute; i.e. $A_{s_i} \cap A_{s_j} \neq \emptyset$ for some i, j. Given a pair of schemas $s_1, s_2 \in \mathcal{S}$, an attribute correspondence is an attribute pair $(a \in A_{s_1}, b \in A_{s_2})$ that reflects the similarity of their semantic meanings. The union of attribute correspondences for all pairs of schemas is denoted as C, each of which is generated by well-known matchers (e.g. COMA++ [11], AMC [12]).

To model a schema group as well as correspondences between attributes of its schemas, we define an affinity matrix $M_{m \times n}$ as an internal data structure. Each of n columns represents an original schema and its attributes. Each of m rows represents an equivalence relation between attributes which belong to different schemas but have similar meanings. More than that, an affinity matrix must satisfy following conditions, which is designed specifically for the schema-reuse problem: (i) each element is either an attribute or null, (ii) all attributes (not null) at column j belongs to the schema $s_j \in \mathcal{S}$, (iii) all attributes (not null) at the same row are equivalent, and (iv) not exists two attributes at different rows are equivalent. Formally, the definition of an affinity matrix is given as follows.

Definition 1 ***Affinity Matrix.*** *Given a schema group $\mathcal{S}$ and a set of correspondences C, an affinity matrix is a matrix $M_{m \times n}$ such that: (i) $\forall i, j : M_{ij} \in A \cup \{null\}$, (ii)*

$\forall i, j : M_{ij} \in A_{s_j}$, *(iii)* $\forall i, j_1 \neq j_2, M_{ij_1} \neq null, M_{ij_2} \neq null : (M_{ij_1}, M_{ij_2}) \in C$, *and (iv)* $\nexists i_1 \neq i_2, j_1, j_2 : (M_{i_1 j_1}, M_{i_2 j_2}) \in C$.

Given a schema group $\mathcal{S}$ and a set of correspondences C, we can construct many affinity matrices, but all of which are just permutations of rows. In order to construct a unique affinity matrix, we define an order relation on $A_{\mathcal{S}}$. We consider each attribute $a \in A_{\mathcal{S}}$ as a character. The order relation between any two attributes is the alphabetical order between two characters. Consequently, a row $M_{i.}$ of the affinity matrix M is a string of n characters. An affinity matrix is *valid* if the string of row r_1 is less than the string of row r_2 ($r_1 < r_2$) with respect to the lexicographical order. Without loss of generality, we hereby consider valid affinity matrices only.

Example 1 *Consider a group of 3 schemas* $\mathcal{S} = \{s_1, s_2, s_3\}$ *and their attributes* $A_{s_1} = \{a_1, a_2\}$, $A_{s_2} = \{b_1, b_2, b_3\}$, $A_{s_3} = \{c_1, c_2, c_3\}$. *We have 8 correspondences* $C = \{(a_1, b_1), (a_1, c_1), (b_1, c_1), (a_2, b_2), (a_2, c_2), (b_2, c_2), (b_3, c_3)\}$. *Assume that the lexicographical order of all attributes is* $a_1 < a_2 < b_1 < b_2 < b_3 < c_1 < c_2 < c_3$. *Then we can construct a unique affinity matrix M for this schema group and these correspondences as in fig. 1.2. The string of the first, second, third row of M is "*$a_1 b_1 c_1$*", "*$a_2 b_2 c_2$*", "*$b_3 c_3$*" respectively. M is valid because "*$a_1 b_1 c_1$*" < "*$a_2 b_2 c_2$*" < "*$b_3 c_3$*" with respect to the assumed lexicographical order.*

In summary, an affinity matrix is an internal data structure that represents a schema group and generated correspondences. Based on the affinity matrix, we can construct anonymized schemas as well as define privacy constraints. These concepts will be described in next subsections. For simplicity sake, the scope of this paper only focuses on one-to-one correspondences. Most well-known matching tools [11, 12] generate this type of correspondence, which relates an attribute of one schema to at most one attribute in another schema. Nonetheless, it must be noted that our model can be extended to one-to-many correspondences (one source attribute to many target attributes) by representing a group of target attributes as a 'complex' attribute [13]. Following this representation, we can treat these correspondences as their one-to-one counter-parts.

3.2 Anonymized Schema

As mentioned above, showing original schemas is dangerous. Although we can hide the identity of schema owners, an adversary can still see the whole schema and make use of external information (e.g. list of possible contributors) to exploit the provenance of presented attributes. Moreover, when the number of relevant schemas in a group is small, the adversary can easily "generate and test" all possible combinations to know which schema belongs to which owner. Therefore, instead of showing original schemas, we unify them into only a single one, namely *anonymized schema*.

An anonymized schema should meet following desirable properties: (i) representation, and (ii) anonymization. First, the anonymized schema has to be representative to

cover the attributes of a schema group. It should show to end-user all possible attributes and the variations of each attribute. By showing these important attributes as well as their variations (i.e. same meaning but different name), the anonymized schema should cover enough information but not overwhelming. Second, the anonymized schema should be opaque enough to prevent adversaries from reconstructing a part or the whole of an original schema. For example in Figure 1.2, given the schema group S_1, S_2, S_3, the anonymized schema should show that there are three distinct attributes $Name$, $CCNum$, and $Expiry$. In that, the attribute $Name$ has two variations $Name$ and $Holder$. Whereas, the attribute $CCNum$ and $Expiry$ have three variations (Num, $CCNum$, CC) and one variation ($Expiry$), respectively.

Our approach is inspired by the generalization technique [14], with some modifications, to protect the owners of original schemas. In this technique, the schemas are combined together such that if a user were to reuse existing schema attributes, he would not know to which owner they belong. One benefit of our approach is that the reconstruction of original schemas from an anonymized schema is extremely hard. The "generate and test" attack is not effective due to the combinatorial explosion of linking each attribute to its schema and each schema to its owner. In fact, there are many other approaches to define an anonymized schema, such as randomization and suppression [15][16]. Randomization is simply adding noisy attributes; while in suppression, the attributes which need to be protected are replaced by special values (e.g. '*'). Using these methods has some disadvantages. Regarding randomization, it might be easy for an adversary to detect noises since each attribute has semantic. For example, we add a noise $Nmea$ for the attribute $Name$, this noisy attribute has no meaning and can be detected. Concerning suppression, replacing an important attribute or schema could lead to loss of information or misleading semantics. For example, if the attribute $CCNum$ is replaced by '*', user would not know attribute $Expiry$ is the expired date of credit card. These are the reasons why we opt for the generalization technique, which is most relevant to our setting, to develop the solution.

Now we introduce how to construct anonymized schemas from a given schema group. To do this, we represent them as internal data structures called affinity instances. An affinity instance $I_{m\times n}$ is constructed from an affinity matrix $M_{m\times n}$ by removing one or many cells; i.e. $\forall i, j : I_{ij} = M_{ij} \vee I_{ij} = null$. Formally, an anonymized schema $\hat{S}$ is constructed from an affinity instance $I_{m\times n}$ as follows.

$$\hat{S} = \{\hat{A}_1, \ldots, \hat{A}_m \mid \hat{A}_i = \bigcup_{1\leq j\leq n} I_{ij}\} \tag{3.1}$$

where $\hat{A}$ is an abstract attribute (a set of attributes at the same row). In other words, an abstract attribute of an anonymized schema is a set of equivalent attributes (i.e. any pair of attributes has a correspondence). Following the running example in Figure 1.2, a possible affinity instance is I_1 and the corresponding anonymized schema is $\hat{S_1} = \{\hat{A'}_1, \hat{A}_2\}$ with $\hat{A'}_1 = \{a_1, b_1\}$ and $\hat{A}_2 = \{a_2, b_2, c_2\}$.

With this definition, the anonymized schema meets two aforementioned properties—representation and anonymization. First, the representation property is satisfied because by showing abstract attributes, users can observe all possible variations to design their own schemas. Second, the anonymization property is satisfied because it is non-trivial to recover the original affinity matrix from a given anonymized schema. In the next section, we will illustrate this property in terms of the probability that an adversary can construct an original schema (column of affinity matrix) from a set of abstract attributes (elements of anonymized schema).

3.3 Privacy Constraint

Privacy constraint is a mean to represent human-understandable policies by mathematical standards. In order to define a privacy constraint, we have to identify two elements:

- *Sensitive information.* In our setting, the sensitive information is attributes of a given schema. We need to protect this information because of two reasons. First, each attribute has different levels of sensitivity (e.g. credit card number is more sensitive than phone number). Second, some attributes are the source of revenue and business strategy and can only be shared under specific conditions.
- *Privacy requirement.* In this paper, the privacy requirement is to prevent adversaries to infer the provenance of sensitive attributes. By knowing which attribute belongs to which schema, an adversary can attack the database system using various techniques such as SQL injection (see Appendix).

To capture these elements, we employ the concept of *presence constraint* that allows contributors to represent their privacy policies in terms of mathematical formulas. A presence constraint defines an upper bound of the probability $Pr(D \in s|\hat{S})$ that an adversary can infer the presence of a set of attributes D in a particular schema s, if he sees an anonymized schema $\hat{S}$. The formal definition of the presence constraint is given as follows.

Definition 2 ***Presence Constraint.*** *A presence constraint γ is a triple $\langle s, D, \theta \rangle$, where s is a schema, D is a set of attributes, and θ is a specified threshold. An anonymized schema $\hat{S}$ satisfies the presence constraint γ if $Pr(D \in s|\hat{S}) \leq \theta$.*

Checking presence constraint. Now we show how to check an anonymized schema satisfying a presence constraint. Given an anonymized schema $\hat{S}$ constructed from the affinity instance I and a presence constraint $\gamma = \langle s, D, \theta \rangle$, we can check whether $\hat{S}$ satisfies γ by computing the probability $Pr(D \in s \mid \hat{S})$ as follows:

$$Pr(D \in s \mid \hat{S}) = \begin{cases} \frac{1}{\prod_{i=1}^{n} |\hat{A}_i|}, & \text{if } \hat{A}_1, \ldots, \hat{A}_k \neq \emptyset \\ 0, & \textit{otherwise} \end{cases} \tag{3.2}$$

where $D = \{a_1, \ldots, a_k\}$ is a subset of attributes of s and $\hat{A}_i \in \hat{S}$ is the abstract attribute (a group of attributes at the same row of I) that contains a_i. In case there is no abstract attribute which contains a_i, we consider $\hat{A}_i = \emptyset$. Note that the presence constraint is often defined on a set of attributes, not one attribute. This is because in real scenarios, leaking information of only one attribute is far less dangerous than that of two or more attributes (e.g. revealing both the credit card number and the security code is more dangerous than revealing only one of them).

Example 2 *Continuing our running example in Fig. 1.2, we consider the anonymized schema* $\hat{S} = \{\hat{A}_1, \hat{A}_2\}$*, where* $\hat{A}_1 = \{a_1, c_1\}$ *and* $\hat{A}_2 = \{a_2, b_2\}$*. Based on eq.* (3.2)*, we have* $Pr(\{a_1, a_2\} \in s_1 \mid \hat{S}) = \frac{1}{|\hat{A}_1| \cdot |\hat{A}_2|} = 0.25$*. Obviously,* $\hat{S}$ *satisfies* $\gamma_1 = \langle s_1, \{a_1, a_2\}, 0.6\rangle$ *but does not satisfies* $\gamma_2 = \langle s_1, \{a_1, a_2\}, 0.1\rangle$*.*

Privacy vs. utility trade-off. It is worth noting that there is a trade-off between privacy and utility [18]. In general, anonymization techniques reduce the utility of querying results when trying to provide privacy protection [19]. On one hand, the utility of an anonymized schema is maximal when there is no presence constraint (i.e. $\theta = 1$). On the other hand, when user set the threshold θ of a presence constraint $\langle s, D, \theta\rangle$ very small (closed to 0), D will not appear in the anonymized schema. Consequently, the utility of this anonymized schema is low since we loss the information of these attributes. As a result, privacy should be seen as a spectrum on which users can choose their place. From now on, we use two terms—*privacy constraint* and *presence constraint*—interchangeably to represent the privacy spectrum defined by users.

4

Anonymized Schema Quality

In this section, we will discuss about the quality of an anonymized schema. Technically, the quality of an anonymized schema $\hat{S}$ is measured by a utility function $u : \overline{\mathbb{S}} \rightarrow \mathbb{R}$, where $\overline{\mathbb{S}}$ is the set of all possible anonymized schemas. The utility value reflects the amount of information $\hat{S}$ carries. To define this utility function, we first introduce two main properties of an anonymized schema in the next two subsections: *importance* and *completeness*.

4.1 Importance

The importance of an anonymized schema is measured by the popularity of attributes it contains. The higher popularity of these attributes, the higher is the importance of this anonymized schema. Technically, we denote the importance of an anonymized schema as a function $\sigma : \bar{S} \rightarrow \mathbb{R}$ and the popularity of an attribute as a function $t : A_{\mathbb{S}} \rightarrow [0, 1]$, where $A_{\mathbb{S}}$ is a set of possible attributes and the domain value of t is normalized to $[0, 1]$ for comparison purposes. There are many aggregate methods (e.g. sum, average) to define $\sigma(.)$ from $t(.)$. In this paper, we compute the importance of a particular anonymized schema $\hat{S}$ by summing over the popularity of all attributes in $\hat{S}$; i.e. $\sigma(\hat{S}) = \sum_{a \in \hat{S}} t(a)$. Consequently, we could say that one anonymized schema is important than another anonymized schema because it has more attributes or its attributes are more popular than those of the other anonymized schema.

We compute the popularity of an attribute $t(.)$ based on two kinds of information, which capture the relationship between attributes and their containing schemas:

- *Cardinality.* The cardinality of an attribute is the number of schemas it appears in. If there are many schemas containing an attribute, then that attribute is likely to be of greater popularity than another one with fewer schemas.
- *Significance.* The significance of a schema is measured by following observations. A schema is significant than another schema if it contains more popular attributes. At the same time, an attribute which belongs to more significant schemas is more popular than those with less significant schemas.

As we attempt to develop a model that combines these two concepts, we notice some similarities to the world-wide-web setting. In this setting, the importance of a web page is determined by both the goodness of the match of search terms on the page itself and the links connecting to it from other important pages. Adapting this idea to our context, we apply the hub-authority model [20] to quantify the popularity of an attribute and the significance of related schemas. The details of this model are described in the Appendix.

4.2 Completeness

The completeness reflects how well an anonymized schema collectively covers the attributes of original schemas, with each abstract attribute is responsible for original attributes it represents. This coverage is necessary to penalize trivial anonymized schemas that contain only one attribute appearing in all original schemas. To motivate the use of this property, we illustrate some observations in Figure 1.2. In this running example, $\hat{S}_2$ has more abstract attributes than $\hat{S}_3$ ($\hat{S}_2$ has three while $\hat{S}_3$ has two). As $\hat{S}_3$ is lack of information about $CCNum$ or CC attributes, users would prefer $\hat{S}_2$ in order to have a better overview of necessary attributes for their schema design. Intuitively, an anonymized schema is more preferred if it covers more information of original schemas.

In this paper, we measure the completeness of an anonymized schema by the number of abstract attributes it contains. The more abstract attributes, the higher is the completeness. Technically, the completeness is defined as a function $\lambda : \bar{S} \rightarrow \mathbb{R}$ and $\lambda(\hat{S}) = |\hat{S}|$. Although the importance of an anonymized schema captures the popularity of covered attributes, it clearly does not reflect how many attributes covered by this anonymized schema. Complementing this aspect, the completeness is practically significant since in schema design, users usually prefer more attributes with less variations rather than less attributes with more variations.

4.3 Put It All Together

As mentioned above, there are two properties (importance and completeness) needs to be considered for the purpose of comparing different anonymized schemas. We attempt to combine these properties into a single comprehensive notion of *utility* that measures the quality of an anonymized schema. Formally, we have:

$$u(\hat{S}) = w \times \sigma(\hat{S}) + (1 - w) \times \lambda(\hat{S}) \tag{4.1}$$

where $w \in [0, 1]$ is a regularization parameter that controls the trade-off between completeness and importance. In terms of comparison, an anonymized schema is more preferred than another anonymized schema if its utility is higher. Back to the running

example in Figure 1.2, consider three anonymized schemas $\hat{S}_1$, $\hat{S}_2$ and $\hat{S}_3$. By definitions, we have $\sigma(\hat{S}_1) = 5$, $\sigma(\hat{S}_2) = \sigma(\hat{S}_3) = 4$ and $\lambda(\hat{S}_1) = \lambda(\hat{S}_2) = 3$, $\lambda(\hat{S}_3) = 2$. Assuming $w = 1$, we have $u(\hat{S}_1) > u(\hat{S}_2) > u(\hat{S}_3)$. $\hat{S}_1$ is more preferred than $\hat{S}_2$ since $\hat{S}_1$ contains more original attributes (importance property). While, $\hat{S}_2$ is more preferred than $\hat{S}_3$ since $\hat{S}_2$ contains more abstract attributes (completeness property).

5

Maximizing Anonymized Schema

Maximizing anonymized schema is the selection of an optimal (w.r.t. utility function) anonymized schema among potential ones that satisfy pre-defined privacy constraints. Formally, the maximization problem of our schema reuse setting is defined as follows.

Problem 1 ***(Maximizing Anonymized Schema)*** *Given a schema group $\mathcal{S}$ and a set of privacy constraints Γ, construct an anonymized schema $\hat{S}$ such that $\hat{S}$ satisfies all constraints Γ, i.e. $\hat{S} \models \Gamma$, and the utility value $u(\hat{S})$ is maximized.*

Now we present an intractability result for the decision version of maximizing anonymized schema, showing that the problem is NP-complete even for a group of only one schema.

Theorem 1 *Let $\mathcal{S}$ be a schema group and Γ be a set of privacy constraints. Then, for a constant U, the problem of determining if there exists an anonymized schema $\hat{S} \models \Gamma$, whose utility value $u(\hat{S})$ is at most U, is NP-complete.*

Proof 1 (Sketch) *To show the decision problem is NP-complete, let us consider a special case in which the schema group contains only one schema and all presence constraints are defined on every pairs of attributes with threshold $\theta = 0$. This special case can be transformed into the* Independent Set *problem, which is known to be NP-complete. The full proof is omitted here for brevity sake and can be found in the Appendix.*

In brief, solving the problem of maximizing anonymized schema requires investigating all permutations of all subsets of correspondences. As such, the space that needs to be explored is exponential in the size of the set of generated correspondences. In the next section, we consider the heuristic-based approach to relax optimization constraints and find the approximate solution in polynomial time.

5.1 Algorithm

In light of Theorem 1, we necessarily consider heuristic approaches to the maximizing anonymized schema problem. Our heuristic-based algorithm takes two parameters as input: a schema group modeled by an affinity matrix M and a set of presence constraints Γ. It will efficiently return an approximate solution—an anonymized schema $\hat{S}$ which satisfies Γ—with the trade-off that the utility value $u(\hat{S})$ is not necessarily maximal. In fact, we found it non-trivial to develop such an algorithm. The key difficulty is that during the optimization process, repairing the approximation solution to satisfy one constraint can break another. At a high level, our algorithm makes use of greedy and local-search heuristics to search the optimal anonymized schema in the space of conflict-free candidates (i.e. anonymized schemas without violations). The details are illustrated in Algorithm 1 and described in what follows.

Algorithm 1: Heuristics-based Algorithm

```
   input : ⟨M, Γ⟩ // An affinity matrix M and a set of predefined constraints Γ
   output: ℑ_op // Maximal Affinity Instance
   denote: u(I^{-a}): utility of I after removing attribute a out of I
           u(I^{+a}): utility of I after adding attribute a into I
 1 A_del = ∅ // Set of deleted attributes; Γ_s = ∅ // Set of satisfied constraints
 2 I = M
 3 while Γ_s ≠ Γ do
 4     /* Violation Repair: greedy deletion until satisfying constraints */
 5     â = argmax_{a∈{I_ij}} u(I^{-a})
 6     I.delete(â); A_del = A_del ∪ {â}
 7     Γ_s = I.checkConstraints()
 8 I_op = I; T = Queue[k] // a queue with fixed size k
 9 while Δ_stop do
10     /* Local search by non-deterministic insertion */
11     Ω = {⟨a, u(I^{+a})⟩ : a ∈ A_del}}
12     â = Ω.rouletteSelection()
13     I.add(â) ; T.add(â)
14     Γ_s = I.checkConstraints()
15     while Γ_s ≠ Γ do
16         /* Violation Repair: greedy deletion until satisfying constraints */
17         â = argmax_{a∈{I_ij}\T} u(I^{-a})
18         I.delete(â)
19         Γ_s = I.checkConstraints()
20     if u(I) > u(I_op) then
21         I_op = I
22 return I_op
```

Technically, we begin by constructing a valid affinity instance from the original affinity matrix (line 2 to line 7). The core idea is we generate an initial affinity instance $I = M$ and then continuously remove the attributes of I until all constraints are satisfied ($\Gamma_s = \Gamma$). The challenge then becomes how to choose which attributes for deletion such that the resulting instance has maximal utility. To overcome this challenge, we use a repair routine (line 3 to line 7) which greedily removes the attributes out of I to eliminate all violations. The greedy choice is that at each stage, we remove the

attribute $\hat{a}$ with lowest utility reduction (line 5); i.e. choose an attribute a such that the utility of I after removing a is maximized. However, this greedy strategy could make unnecessary deletions since some constraints might share common attributes, thus making the utility not maximal. For example in Figure 1.2, consider the anonymized schema $\hat{S} = \{\{a_1, c_1\}, \{a_2, b_2\}\}$ with two predefined constraints $\gamma_1 = Pr(a_1 \in s_1 \mid \hat{S}) \leq 0.4$ and $\gamma_2 = Pr(\{a_1, a_2\} \in s_1 \mid \hat{S}) \leq 0.1$. The best way to satisfy these constraints is removing only attribute a_2, but the algorithm will delete a_1 first and then a_2. Hence, we need to insert a_2 back into $\hat{S}$ to increase the utility. For this purpose, A_{del} stores all deleted attributes (line 6) for the following step.

The above observation motivates the next step in which we will explore neighbor instances using local search and keep track of the instance with highest utility (line 9 to line 21). Starting with the lower-bound instance from the previous step ($I_{op} = I$), the local search is repeated until the termination condition Δ_{stop} is satisfied (e.g. k iterations or time-out). In each iteration, we incrementally increase the utility of the current instance I by reverting unnecessary deletions from the previous step. The raising issue is how to avoid local optima if we just add an attribute with highest utility gain; i.e. choose an attribute a such that the utility of I is maximized after adding a. To tackle this issue, we perform two routines:

- *Insertion.* Each attribute $a \in A_{del}$ is a possible candidate to insert into I (line 11). We use Roulette-wheel selection [21] as a non-deterministic strategy, where an attribute with greater utility has higher probability to add first (line 12).
- *Repair.* When a particular attribute is inserted, it might make some constraints unsatisfied ($\Gamma_s \neq \Gamma$). Thus, the repair routine is invoked again to eliminate new violations by removing the potential attributes out of I (line 15 to line 19).

In an iteration of local-search, an attribute could be inserted into an affinity instance and then removed immediately by the repair routine, making this instance unchanged and leading to being trapped in local optima. For this reason, we employ the Tabu search method [22] that uses a fixed-size "tabu" (forbidden) list T of attributes so that the algorithm does not consider these attributes repeatedly. In the end, the maximal affinity instance I_{op} is returned by selecting the one with the highest utility among explored instances (line 21).

5.2 Algorithm Analysis

Our algorithm is terminated and correct. Termination follows from the fact that the stopping condition Δ_{stop} can be defined to execute at most k iterations (k is a tuning parameter). The correctness follows from the following points. (1) When a new attribute a added into I (line 13) violates some constraints, I is repaired immediately. This repair routine takes at most $\mathcal{O}(|M|)$, where $|M|$ is the number of attributes of the schema group represented by M. (2) The newly added attributes are not removed in the repair routine since they are kept in the "tabu" list T. The generation of Ω takes at most $\mathcal{O}(|M|)$. (3) I_{op} always maintains the instance with maximal utility (line 21).

Sum it up, the worse-case running time is $\mathcal{O}(k \times |M|^2)$—which is reasonably tractable for real datasets.

Looking ahead for repair cost. We have another approach to improve the repair technique. The goal of this approach is to avoid bad repairs (that cause many deletions) by adding some degree of lookahead to the repair cost (i.e. number of deletions). In Algorithm 1, the utility value $u(I^{-a})$ is used as an indicator to select an attribute for deletion. Now we modify this utility value to include a look-ahead approximation of the cost of deleting one further attribute. More precisely, when an attribute a is considered for deletion (first step), we will consider a next attribute a' given that a is already deleted (second step). Our look-ahead function will be the utility of the resulting instance after these two steps. In other words, we prevent a bad repair by computing the utility of the resulting instances one step further.

5.3 Improving Performance

After analyzing the complexity of our algorithm, we attempt to improve its performance by using two important heuristics.

Redundant Computation. We avoid redundant computations of checking privacy constraints by tracking unsatisfied constraints. The rationale behind this heuristic is that a modification of instance (adding or removing one attribute) only affects a few constraints. Thus, it is imprudent to re-check all constraints per modification. Technically, we propose an important efficiency optimization by keeping track of all unsatisfied constraints for each attribute. To accomplish this, we maintain an internal data structure $\langle a, \Gamma_a \rangle$ which map each attribute a to a set of constraints $\Gamma_a \subseteq \Gamma$ in which a involved. Denote Γ_u is a set of unsatisfied constraints. Specially, we improve the function *checkConstraints*() by following maintenance mechanism: (i) *Initialization* - for each attribute a, Γ_a initially contains all constraints that involve a, Γ_u is initialized as empty. (ii) *Each repair step* - when an attribute a is deleted from the current instance I, new violated constraints in Γ_s moved to Γ_u and then all constraints Γ_a are removed out of Γ_u. (iii) *Each insertion step* - when an attribute a is inserted to the current instance I, re-check all constraints in Γ_a then these satisfied and unsatisfied constraints are added into Γ_s and Γ_u respectively.

This maintenance is efficient due to the following observations. In the repair routine, the rule (ii) works since a constraint violation will become valid if any attribute involved in the violation is removed. However, this observation is not totally valid for the insertion routine. When we insert an attribute a, not all constraints related to a are violated, depending on the existence of other involved attributes. Although we need to re-check these constraints Γ_a, this computation is several orders of magnitude faster the checking of all constraints Γ since $|\Gamma_a| \ll |\Gamma|$. As a result, this optimization is practically important and always used in our experiments.

Constraint Decomposition. The key challenge in the maximizing anonymized schema problem is satisfying one constraint (through deletion and insertion of attributes) can break other constraints. In other words, two constraints are independent of each other if satisfying one of them does not affect the other. To reduce the complexity, we decompose the set of presence constraints into inter-independent subsets, called *conflicting zones* in which any two constraints in two different zones are independent of each other. Technically, let D_1 and D_2 are the sets of attributes involved in the constraint γ_1 and γ_2, respectively. Let M be an affinity matrix and M_r is a set of all attributes at the row r. Two constraints γ_1 and γ_2 are dependent iff exists a row r of M such that $D_1 \cap M_r \neq \emptyset$ and $D_2 \cap M_r \neq \emptyset$. Based on this relationship, we build a graph $G = (V, E)$, where a vertex $v \in V$ is a constraint, an edge $e = (v_i, v_j) \in E$ indicates the dependence between v_i and v_j. A conflicting zone z is a connected component of G. Finding the set of all connected components can be computed efficiently in $O(|V| + |E|)$ time.

Now we show that constraint decomposition significantly reduces the complexity of our problem. The original problem is now divided into sub-problems—each of which is the maximizing anonymized schema problem for a given zone z. We denote $|z|$ is the number of attributes in this zone; i.e. $|z| = |\bigcup D_i : \gamma_i \in z|$. One can imagine that the naive approach of a sub-problem is enumerating all possible subsets of attributes, the number of which is $2^{|z|}$. Without partitioning, the cost of finding a solution for the original problem is $\mathcal{O}(2^{|z_1|+\ldots+|z_k|})$. With partitioning, we simply combine the solutions of sub-problems to construct that solution in $\mathcal{O}(2^{|z_1|} + \ldots + 2^{|z_k|})$ time. Since $\mathcal{O}(2^{|z_1|} + \ldots + 2^{|z_k|}) << \mathcal{O}(2^{|z_1|+\ldots+|z_k|})$, this optimization is significantly useful and empirically studied in our experiments.

6

Experiments

The main goal of the following evaluation is to examine our approach about schema reuse with privacy constraints. To verify the effectiveness of the proposed framework in designing and constructing the anonymized schema, following experiments are performed: (i) effects of different heuristics for the maximizing anonymized schema problem, (ii) the correctness of utility function, (iii) the computation time of our heuristic-based algorithm in various settings, and (iv) the tradeoff between utility and privacy. We proceed to report the results on real datasets and synthetic data.

6.1 Experimental Settings

Datasets. Our experiments will be conducted on two types of data: real data and synthetic data. While the real data provides a pragmatic view on real-world scenarios, the synthetic data helps to evaluate the performance with different settings.

- *Real Data.* For evaluation purposes, we only examine well-known datasets for which the sources are available and licensing permits their use. We have used 3 real-world datasets, extracted from various domains: Google Fusion Tables, UniversityAppForm, and WebForm. They are publicly available on our website [1] for repeatability and extensibility.
- *Synthetic Data.* In order to have the flexibility of controlling many parameters (input size, privacy threshold, number of constraints etc.), we also use synthetic data. We want to simulate practical cases in which one attribute is used repeatedly in different schemas (e.g. 'username'). To generate a set of n synthetic schemas, two steps are performed. In the first step, we generate k core schemas, each of which has m attributes, such that all attributes within single schema as well as across multiple schemas are completely different. Without loss of generality, for any two schemas a and b, we create all 1-1 correspondences between their

[1] `http://lsirwww.epfl.ch/schema_matching`

attributes; i.e. $(a_i, b_i)_{1 \leq i \leq m}$. In the second step, $n-k$ remaining schemas are permuted from k previous ones. For each schema, we generate m attributes, each of which is copied randomly from one of k attributes which have all correspondences between them; i.e. attributes at the same row of the affinity matrix.

Privacy Constraints. To generate a set of privacy constraints Γ for simulation, we need to consider two aspects:

- *The number of attributes per constraint.* To mix the constraints with different size, we set a parameter α_i as the number of constraints with i attributes. Since the chance for an adversary to conclude a group of four or more attributes is small, we only consider constraints with less than four attributes (i.e. $\alpha_{i \geq 4} = 0$). For all experiments, we set $\alpha_1 = 50\%$, $\alpha_2 = 30\%$, and $\alpha_3 = 20\%$.
- *Privacy threshold.* We generate the privacy threshold θ of all constraints according to uniform distribution in the range $[a, b]$, $0 \leq a, b \leq 1$. In all experiments, we set $a = (1/n)^i$ and $b = (2/n)^i$, where n is the number of schemas and i is the number of attributes in the associated constraint.

In general, the number of constraints is proportional to the percentage of the total number of attributes; i.e. $|\Gamma| \propto m \cdot n$). In each experiment, we will vary the percentage value differently.

Evaluation Metrics. Beside measuring the computation time to evaluate proposed heuristics, we also consider two important metrics:

- *Privacy Loss.* Privacy loss measures the amount of disagreement between the two probability distributions: actual privacy and expected privacy. Technically, given an anonymized schema $\hat{S}$ and a set of presence constraints $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ where $\gamma_i = \langle s_i, D_i, \theta_i \rangle$, we denote $p_i = Pr(D_i \in s_i | \hat{S})$. Privacy loss is computed by the Kullback-Leiber divergence between two distributions $P = \{p_i\}$ (actual) and $\Theta = \{\theta_i\}$ (expected):
$$\Delta p = KL(P \parallel \Theta) = \sum_i p_i \log \frac{p_i}{\theta_i} \tag{6.1}$$
- *Utility Loss.* Utility loss measures the amount of utility reduction with regard to the existence of privacy constraints. Technically, denote $u_\emptyset$ is the utility of an anonymized schema without privacy constraints and u_Γ is the utility of an anonymized schema with privacy constraints Γ, utility loss is calculated as:
$$\Delta u = \frac{u_\emptyset - u_\Gamma}{u_\emptyset} \tag{6.2}$$

We implement our schema reuse framework in Java and use it to evaluate the effectiveness of our approach. All the experiments ran on an Intel Core i7 processor 2.8 GHz system with 4 GB of RAM.

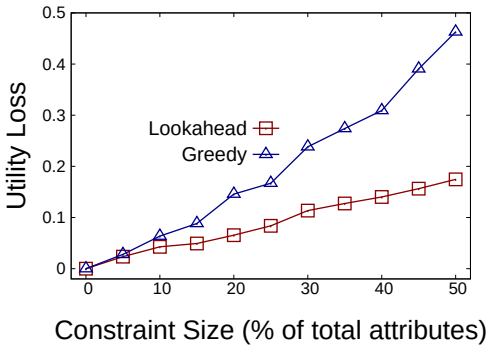


Figure 6.1: Utility Loss (the lower, the better)

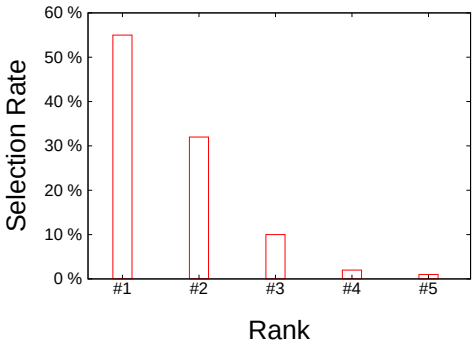


Figure 6.2: Correctness of Utility Function

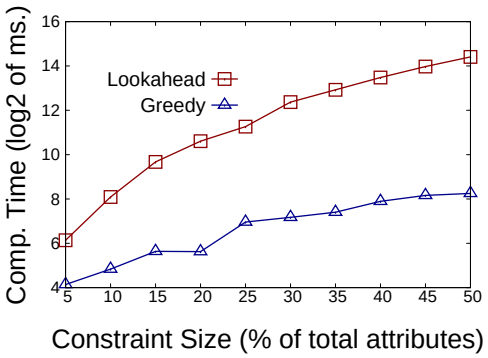


Figure 6.3: Comp. Time: vary #constraints

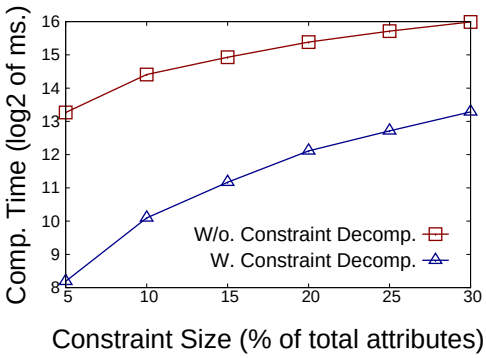


Figure 6.4: Comp. Time: effects of constraint decomp.

6.2 Effects of Heuristics

This experiment will compare the utility loss of the anonymized schema returned by different heuristics. As aforementioned in Section 5, the maximizing anonymized schema problem is NP-Complete and a heuristic-based algorithm is proposed. In this experiment, we study the proposed algorithm with two heuristics, namely: (1) Greedy (without lookahead): choose candidate attributes with the original utility function, and (2) Lookahead (greedy with lookahead): choose candidate attributes by using the modified utility function with looking ahead of utility change. We use a synthetic data with a random number of schemas $n \in [50, 100]$ and each schema has a random number of attributes $m \in [50, 100]$. In that, the copy percentage is varied from 10% to 20%; i.e. $k/n \in [0.1, 0.2]$. The number of constraints is varied from 5% to 50% of total number of attributes. The results for each configuration are averaged over 100 runs. Regarding utility function, we set $w = 1$ which means the completeness is equally preferred as the

importance.

Figure 6.1 depicts the result of two described heuristics. The X-axis shows the number of constraints, while the Y-axis shows the utility loss (the lower, the better). A noticeable observation is that the Lookahead heuristic performs better than the Greedy heuristic. Moreover, when there are more constraints, this difference is more significant (up to 30% when the number of constraints is 50%). This is because the Lookahead heuristic enables our algorithm to foresee and avoid bad repairs, which cause new violations when eliminating the old ones, thus improves the quality of the result.

6.3 Correctness of Utility Function

In this experiment, we would like to validate the correctness of our utility function by comparing the results of the proposed algorithm with the choices of users. Technically, we expect that the ranking of anonymized schemas by utility function is equivalent to their ranking by users. Regarding the setting, we use the real data to extract 20 different schema groups, each of which is constructed by extracting random numbers of schemas and attributes. For each group, we present five of constructed anonymized schemas (ranked $\#1, \#2, \#3, \#4, \#5$ by the decreasing order of utility) to 10 users (users do not know the order). These presented anonymized schemas are not top-5, but taken from the highest such that the utility difference between two consecutive ranks is at least 10% to avoid insignificant differences. Each user is asked to choose the best anonymized schema in his opinion (200 votes in total). We fix the number of constraints at 30% of total attributes. Since the Lookahead strategy is better than the Greedy strategy as presented in the previous experiment (Section 6.2), we use it to construct anonymized schemas.

The results are depicted in Figure 6.2. X-axis shows top-5 anonymized schemas returned by our algorithm. Y-axis presents the selection rate (percentage of users voting for an anonymized schema), averaged over all schema groups. A key finding is that the order of selection rate corresponds to the ranking of solutions by the utility function. That is, the first-rank anonymized schema (#1) has highest selection rate (55%) and the last-rank anonymized schema (#5) has lowest selection rate (1%). Moreover, most of users opt for the first-rank and second-rank solution (88% in total), indicating that our algorithm is able to produce anonymized schemas similar to what users want in real-world datasets.

6.4 Computation Time

Various schema-reuse applications often have limited resources on computing speed, or pre-defined timeouts on searching queries. As a result, the computation time to construct an anonymized schema becomes an important aspect, when we consider different heuristics. In this experiment, we design various settings to study this aspect

with two above-mentioned strategies (Greedy vs. Lookahead) using synthetic data. In each setting, we measure the average computation time over 100 runs.

Effects of Input Size. In this setting, the computation time is recorded by varying the input size of the affinity matrix; i.e. the number of schemas and the number of attributes per schema. Table 6.1 shows the output of this setting. A significant observation is that the Greedy strategy outperforms the Lookahead strategy. For example with the largest input size (50×100), while the Greedy strategy returns the result less than 7 seconds, the Lookahead strategy exceeds 3000 seconds, which is not suitable for applications that require fast response time. This is because the cost of looking-ahead utility change is expensive with the trade-off that the utility loss is smaller as presented in the previous experiment (Section 6.2). We recommend interested readers to use this result as a guideline to choose the best-suited heuristics for their specific applications.

Effects of Constraints Size. In this setting, the computation time is recorded by varying #constraints from 5% to 50% of total number of attributes. Based on results of the previous setting, we fix the input size at 20×50 for a low starting point. Figure 6.3 illustrates the output, in which the X-axis is the constraint size and the Y-axis is the computation time (ms.) in logarithmic scale of base 2.

A noticeable observation is that the Lookahead strategy is much slower than the Greedy strategy. For example, with #constraints of only 5%, the Greedy strategy is already 4-times faster than the Lookahead strategy and this difference is larger with more constraints. In fact, adding degrees of look-ahead into utility function actually degrades the performance. The running time of those extra computations increases due to the large number of combinations of privacy constraints. Like the previous setting, potential applications should make use of this result to choose the best strategy accordingly, in terms of both utility and computation time.

Effects of Constraint Decomposition. Now we study the impact of constraint decomposition on the computation time. In this setting, we report only the Greedy strategy (the Lookahead strategy is omitted due to similar behaviors). The computation time is compared between two cases: (i) Greedy strategy without constraint decomposition, and (ii) Greedy strategy with constraint decomposition. In the former case, the Greedy strategy is applied on the original problem; while in the latter case, we

Table 6.1: Computation Time (log2 of msec.)

Size of M *	Greedy	LookAhead
10×20	2.58	5.04
10×50	5.13	10.29
20×50	7.30	13.36
20×100	9.95	17.60
50×100	12.62	21.58

* $m \times n$: m attributes (per schema) and n schemas

divide the original problem into independent sub-problems (as described in Section 5.3) and then apply the Greedy strategy sequentially. The running time of combining solutions of all sub-problems is not counted since it is much smaller than the computation time of each solution itself.

The results are presented in Figure 6.4.The X and Y axes corresponds to the number of constraints (varied from 5% to 30% of total attributes) and computation time, respectively. One can observe that constraint decomposition dramatically reduces the computation time. This is compatible with results of the previous setting, in which the computation time is significantly smaller with fewer constraints. Moreover, the constraint decomposition also enables us (in future work) to extend proposed heuristics with parallel computing, which is especially useful when the problem size is much larger.

6.5 Trade-off between Privacy and Utility

In this experiment, we validate the trade-off between privacy and utility. As pointed out in Section 3.3, the utility of the resulting anonymized schema reduces when we enforce high privacy. Thus, we gave users a parameter (i.e. presence threshold θ) to specify the privacy according to their expectations. To support users decide on the appropriate value of θ, we examine its effects. More precisely, we relax each presence constraint $\gamma \in \Gamma$ by increasing the threshold θ to $(1 + r)\theta$, where r is called relaxing ratio and varied according to normal distribution $\mathcal{N}(0.3, 0.2)$. Based on previous experiments, we use the Lookahead strategy on the synthetic data with different input size and fix the number of constraints at 30% of total attributes. For a resulting anonymized schema, privacy is quantified by privacy loss (eq. (6.1)), while utility is quantified by utility loss (eq. (6.2)). For comparison purposes, both utility loss and privacy loss values are normalized to $[0, 1]$ by the thresholding technique: $\Delta u = \frac{\Delta u - min_{\Delta u}}{max_{\Delta u} - min_{\Delta u}}$ and $\Delta p = \frac{\Delta p - min_{\Delta p}}{max_{\Delta p} - min_{\Delta p}}$, where $max_{\Delta u}$, $min_{\Delta u}$, $max_{\Delta p}$, $min_{\Delta p}$ are the maximum and minimum values of utility loss and privacy loss in this experiment.

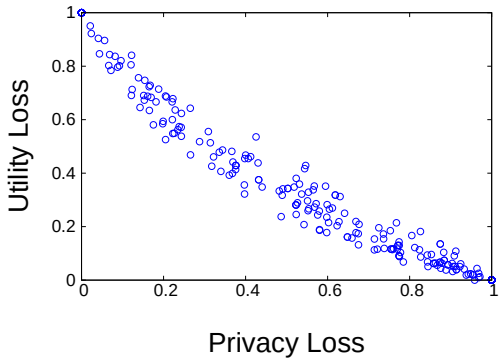


Figure 6.5: Trade-off between privacy and utility (lower, better)

Figure 6.5 shows the distribution of all the points whose x and y values are privacy loss and utility loss, respectively. Each point represents the resulting anonymized schema in a specific configuration of the relaxing ratio and the input size. One can easily observe a trade-off between privacy and utility: the utility loss decreases when the privacy loss increases and vice-versa. For example, the maximal privacy (privacy loss = 0) is achieved when the utility is minimal (utility loss = 1). This is because the decrease of presence threshold θ will reduce the number of presented attributes, which contributes to the utility of resulting anonymized schemas.

7

Related Work

We now review salient work in schema reuse, privacy models, anonymization techniques, and utility measures that is related to our research. Some concepts are briefly interpreted in our setting.

Schema reuse. There is a large body of research on schema-reuse. Some of the works focus on building large-scale schema repositories [6, 8, 23, 24], designing management models to maintain them [25, 26], and establishing semantic interoperability between collected schemas in those repositories [11, 27, 28]. While, some of the others support to find relevant schemas according to a user query, including similarity metrics [9, 10, 29], techniques to speed-up the querying process by clustering relevant schemas into multiple domains [30], and visualization aspects [31, 32]. In the context of reusing information, there is another approach that reuses the design of databases through ERD models [33]. This approach is out of the scope of this paper. Unlike these works, we aim at reusing schemas with privacy constraints for the purpose of encouraging contributors to share their schemas.

Privacy Models. Privacy model is a mean to prevent information disclosure by representing user-defined policies under mathematical standards. Numerous types of information disclosure have been studied in the literature [34], such as attribute disclosure [35] and identity disclosure [36], and membership disclosure [37]. In the context of schema reuse, the linking of attribute to its containing schema is considered as an information disclosure. There is a wide body of work have proposed privacy models, including k-anonymity [38], l-diversity [39], and t-closeness [40]. These models can be interpreted in the schema-reuse setting as follows. The k-anonymity model prevents identity disclosure by requiring each attribute is indistinguishable from at least $k-1$ other attributes which have correspondences to it. The l-diversity model prevents attribute disclosure by requiring that in any group of sensitive attributes that have correspondences between them, there are l distinct attributes. The t-closeness model extends the l-diversity one by allowing different groups of sensitive attributes to have different l values. Similar to most of these works, the model proposed in this paper employs the concept of presence constraint. In that, we define a presence threshold θ

which provides flexibility to limit the chance that an adversary can infer the linking of a sensitive attribute to its containing schema; i.e. the probability of inferring a sensitive attribute is bounded by different user-defined threshold.

Anonymization Techniques To implement privacy models, there are a wide range of techniques have been proposed in the literature. Five of the most popular techniques are generalization, bucketization, randomization, suppression, and differential privacy. In terms of schema reuse, these techniques can be interpreted as follows. The generalization technique [14] replaces the sensitive attribute with a common attribute, which still preserves the utility of attributes shown to users (no noises). The bucketization technique [41] extends the generalization technique for multi-dimensional data through partitioning. The randomization technique [15, 42] adds a "noisy" attribute (e.g. 'aaa'). The suppression technique [16] replaces a sensitive attribute by a special value (e.g. '*'). The differential privacy technique [36] enhances both generalization and suppression techniques by balancing between preserving the utility of attributes and adding noises to protect them. Similar to most of the existing research, we adopt the generalization technique (which does not add noises) to preserve the utility of attributes shown to users. This utility is important in schema design since adding noisy attributes could lead to data inconsistencies.

Quality Measures. Recent research has proposed various types of measures to capture the quality in data integration. Authors of [43, 44] dealt with the quality of data source or querying results. While, the work in [45] concerned about the quality of the mediated schema, which provides a unified querying interface of data sources. Similar to [45], our work focuses on the quality of an anonymized schema that is also a unification of original schemas. To quantify the quality of schema unification, researchers have proposed different quantitative metrics, the most important ones among which are completeness and minimality [46, 47]. While the former represents the percentage of original attributes covered by the unified schema, the latter ensures that no redundant attribute appears in the unified schema. Regarding the quality of a schema attribute, [48] also proposed a metric to determine whether it should appear in schema unification. Combining these works and applying their insights in the schema-reuse context, this paper proposed a comprehensive notion of *utility* that measures the quality of an anonymized schema by capturing how many important attributes it contains and how well it covers a wide range of attribute variations.

8

Conclusions

In this work, we addressed the challenge of preserving privacy when integrating and reusing schemas in schema-reuse systems. To overcome this challenge, we introduced a framework for enabling schema reuse with privacy constraints. Starting with a generic model, we introduce the concepts of schema group, anonymized schema and privacy constraint. Based on this model, we described utility function to measure the amount of information of an anonymized schema. After that, we formulated privacy-preserving schema reuse as a maximization problem and proposed a heuristic-based approach to find the maximal anonymized schema efficiently. In that, we investigated various heuristics (greedy repair, look-ahead, constraint decomposition) to improve the computation time of the proposed algorithm as well as the utility of the resulting anonymized schema. Finally, we presented an empirical study that corroborates the efficiency of our framework with main results: effects of heuristics, the correctness of utility function, the guideline of computation time, and the trade-off between privacy and utility.

Our work opens up several future research directions. One pragmatic direction is to support a "pay-as-you-use" fashion in schema-reuse systems. In that, reuse should be defined at a level finer than complete schemas, as schemas are often composed of meaningful building blocks. Moreover, our techniques can be applied to other domains such as business process and component-based design.

References

[1] http://schema.org/. 1

[2] http://www.factual.com/. 1

[3] http://www.socrata.com/. 1

[4] https://www.niem.gov/. 1

[5] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. **Freebase: a collaboratively created graph database for structuring human knowledge**. In *SIGMOD*, pages 1247–1250, 2008. 1

[6] Hector Gonzalez, Alon Y. Halevy, Christian S. Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, Warren Shen, and Jonathan Goldberg-Kidon. **Google fusion tables: web-centered data management and collaboration**. In *SIGMOD*, pages 1061–1066, 2010. 1, 25

[7] Mehdi Bentounsi, Salima Benbernou, Cheikh S. Deme, and Mikhail J. Atallah. **Anonyfrag: an anonymization-based approach for privacy-preserving BPaaS**. In *Cloud-I*, pages 9:1–9:8, 2012. 1

[8] Kuang Chen, Akshay Kannan, Jayant Madhavan, and Alon Halevy. **Exploring schema repositories with schemr**. *SIGMOD Rec.*, pages 11–16, 2011. 1, 3, 25

[9] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. **Finding related tables**. In *SIGMOD*, pages 817–828, 2012. 1, 3, 25

[10] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. **Annotating and searching web tables using entities, types and relationships**. In *VLDB*, pages 1338–1347, 2010. 1, 3, 25

[11] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. **Schema and ontology matching with COMA++**. In *SIGMOD*, pages 906–908, 2005. 5, 6, 25

[12] Eric Peukert, Julian Eberius, and Erhard Rahm. **AMC - A framework for modelling and comparing matching systems as matching processes**. In *ICDE*, pages 1304–1307, 2011. 5, 6

[13] Avigdor Gal, Tomer Sagi, Matthias Weidlich, Eliezer Levy, Victor Shafran, Zoltán Miklós, and Nguyen Quoc Viet Hung. **Making sense of top-k matchings: a unified match graph for schema matching**. In *IIWeb*, pages 6:1–6:6, 2012. 6

[14] Roberto J Bayardo and Rakesh Agrawal. **Data privacy through optimal k-anonymization**. In *ICDE*, pages 217–228, 2005. 7, 26

[15] Nabil R. Adam and John C. Worthmann. **Security-control methods for statistical databases: a comparative study**. *CSUR*, pages 515–556, 1989. 7, 26

[16] Vijay S Iyengar. **Transforming data to satisfy privacy constraints**. In *SIGKDD*, pages 279–288, 2002. 7, 26

[17] WG Halfond, Jeremy Viegas, and Alessandro Orso. **A classification of SQL-injection attacks and countermeasures**. In *IEEE*, pages 65–81, 2006. 32

[18] Tiancheng Li and Ninghui Li. **On the tradeoff between privacy and utility in data publishing**. In *SIGKDD*, pages 517–526, 2009. 9

[19] Justin Brickell and Vitaly Shmatikov. **The cost of privacy: destruction of data-mining utility in anonymized data publishing**. In *KDD*, pages 70–78, 2008. 9

[20] Jon M Kleinberg. **Authoritative sources in a hyperlinked environment**. *JACM*, pages 604–632, 1999. 11, 31

[21] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989. 15

[22] Fred Glover and Claude McMillan. **The general employee scheduling problem: an integration of MS and AI**. *COR*, pages 563–573, 1986. 15

[23] Jayant Madhavan, P.A. Bernstein, An-Hai Doan, and Alon Y. Halevy. **Corpus-based schema matching**. *ICDE*, pages 57–68, 2005. 25

[24] Kenneth P. Smith, Peter Mork, Len Seligman, Peter S. Leveille, Beth Yost, Maya Hao Li, and Chris Wolf. **Unity: Speeding the creation of community vocabularies for information integration and reuse**. In *IRI*, pages 129–135, 2011. 25

[25] Michael Franklin, Alon Halevy, and David Maier. **From databases to dataspaces: a new abstraction for information management**. *SIGMOD Rec.*, pages 27–33, 2005. 25

[26] Philip A. Bernstein and Sergey Melnik. **Model management 2.0: manipulating richer mappings**. In *SIGMOD*, pages 1–12, 2007. 25

[27] P.A. Bernstein, J. Madhavan, and Erhard Rahm. **Generic Schema Matching, Ten Years Later**. In *VLDB*, pages 695–701, 2011. 25

[28] A. Das Sarma, X. Dong, and Alon Halevy. **Bootstrapping Pay-As-You-Go Data Integration Systems**. In *SIGMOD*, pages 861–874, 2008. 25

[29] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. **WebTables: exploring the power of tables on the web**. In *VLDB*, pages 538–549, 2008. 25

[30] Hatem A. Mahmoud and Ashraf Aboulnaga. **Schema clustering and retrieval for multi-domain pay-as-you-go data integration systems**. In *SIGMOD*, pages 411–422, 2010. 25

[31] Beth Yost, Craig Bonaceto, Michael Morse, Chris Wolf, and Ken Smith. **Visualizing Schema Clusters for Agile Information Sharing**. In *InfoVis*, pages 5–6, 2009. 25

[32] Ken Smith, Craig Bonaceto, Chris Wolf, Beth Yost, Michael Morse, Peter Mork, and Doug Burdick. **Exploring schema similarity at multiple resolutions**. In *SIGMOD*, pages 1179–1182, 2010. 25

[33] Stefania Leone and MC Norrie. **Building eCommerce systems from shared micro-schemas**. In *OTM*, pages 284–301, 2011. 25

[34] Diane Lambert. **Measures of disclosure risk and harm**. *JOS*, pages 313–313, 1993. 25

[35] George T Duncan and Diane Lambert. **Disclosure-limited data dissemination**. *JASA*, pages 10–18, 1986. 25

[36] Cynthia Dwork. **Differential privacy**. In *ICALP*, pages 1–12. 2006. 25, 26

[37] Mehmet Ercan Nergiz, Maurizio Atzori, and Chris Clifton. **Hiding the presence of individuals from shared databases**. In *SIGMOD*, pages 665–676, 2007. 25

[38] Latanya Sweeney. **k-anonymity: a model for protecting privacy**. *IJUFKS*, pages 557–570, 2002. 25

[39] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. **L-diversity: Privacy beyond k-anonymity**. *TKDD*, pages 24–24, 2007. 25

[40] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. **t-closeness: Privacy beyond k-anonymity and l-diversity**. In *ICDE*, pages 106–115, 2007. 25

[41] Xiaokui Xiao and Yufei Tao. **Anatomy: Simple and effective privacy preservation**. In *VLDB*, pages 139–150, 2006. 26

[42] Rakesh Agrawal and Ramakrishnan Srikant. **Privacy-preserving data mining**. *SIGMOD Rec.*, pages 439–450, 2000. 26

[43] Felix Naumann, Ulf Leser, and Johann Christoph Freytag. **Quality-driven integration of heterogeneous information systems**. In *VLDB*, pages 447–458, 1999. 26

[44] Felix Naumann, Johann-Christoph Freytag, and Ulf Leser. **Completeness of integrated information sources**. *ISJ*, pages 583–615, 2004. 26

[45] Petia Assenova and Paul Johannesson. **Improving quality in conceptual modelling by the use of schema transformations**. In *ER*, pages 277–291, 1996. 26

[46] M.C.M. Batista and A.C. Salgado. **Information Quality Measurement in Data Integration Schemas**. In *QDB*, pages 61–72, 2007. 26

[47] Fabien Duchateau and Zohra Bellahsene. **Measuring the quality of an integrated schema**. In *ER*, pages 261–273, 2010. 26

[48] Cong Yu and HV Jagadish. **Schema summarization**. In *VLDB*, pages 319–330, 2006. 26

Appendix

A. Proof for Theorem 1

Given an anonymized schema, it it trivial to check whether it satisfies all constraints Γ (eq. (3.2)) and whether its utility score (eq. (4.1)) is less than U. So the problem clearly is in NP. Now to show it is NP-hard, consider the following simplified version of our problem. Let us have only one schema s in the repository; i.e. $\mathcal{S} = \{s\}$. The constraint set Γ only contains one type of constraint which is defined on a pair of attributes. This constraint compels that the two given attributes cannot appear together; i.e. $Pr(\{a_i \in s, a_j \in s\}|\hat{S}) = 0$. Formally, $\Gamma = \{\gamma|\gamma = Pr(\{a_i, a_j s\}|\hat{S}) = 0\}$, for some pairs $a_i, a_j \in s$. This version can be transformed into the *Independent Set* problem. In that, we can construct an undirected graph $G = (V, E)$ where each vertex $v \in V$ is an attribute and each edge $e = (v_i, v_j) \in E$ represents the constraint between two attributes a_i and a_j. The utility function $u(\hat{S})$ is proportional to the number of attributes in $\hat{S}$. Thus finding $\hat{S}$ with maximal utility is equivalent to finding an independent subset of V with maximal cardinality. Therefore, our problem is NP-complete, since its special case is Independence Set problem, which is known to be NP-complete.

B. Hub-Authority Model for Measuring Schema Importance

As aforementioned in Section 4, we compute the importance of an anonymized schema as $\sigma(\hat{S}) = \sum_{a \in \hat{S}} t(a)$, where $t(.)$ is the popularity of an attribute. We apply the hub-authority model [20] to quantify the popularity of original attributes $t(.)$ as well as the importance of their schemas, denoted as a function $I : \mathcal{S} \rightarrow \mathbb{R}$. The main idea is that a schema is important than another schema if it contains more popular attributes, and an attribute is more popular than another attribute if it belongs to more significant schemas.

Technically, this model represents the relationship between attributes and schemas in a bipartite graph, as illustrated in Figure 8.1. A schema (hub) is connected to its attributes (authorities). An attribute is connected to all schemas that have any attribute corresponding to it. On top of this representation, a mutual recursion is

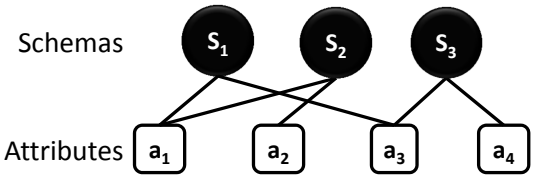


Figure 8.1: Hub-Authority Model. Authority Update Rule: $t(a) = \sum_{s \in \{s \in \mathcal{S} | a \leftrightarrow s\}} I(s)$. Hub Update Rule: $I(s) = \sum_{a \leftrightarrow s} t(a)$.

performed through a series of iterations until convergence, including the authority-update rule and the hub-update rule. For example, if we initialize hub values as 1 and authority values as 1, after the first iteration we have (without normalization) $t(a_1) \leftarrow I(s_1) + I(s_2) = 2$, $t(a_2) \leftarrow I(s_2) = 1$, $t(a_3) \leftarrow I(s_1) + I(s_3) = 2$, $t(a_4) \leftarrow I(s_3) = 1$ and $I(s_1) \leftarrow t(a_1) + t(a_3) = 2$, $I(s_2) \leftarrow t(a_2) = 1$, $I(s_3) \leftarrow t(a_3) + t(a_4) = 2$. And so on, these computations are repeated until convergence.

C. SQL Injection

SQL injection is a code injection technique, used to attack SQL databases, in which malicious SQL statements are inserted into an entry field for accessing the database contents without sufficient privileges (e.g. through the login form of a website) [17]. It is more dangerous if the attacker knows the meta-data (i.e. schemas), such as attribute names. For example, an attacker can inject the statement 'UNION SELECT cardNum, sCode, cardExp from creditCards – ' into the login field, which produces the following query:

```
SELECT * FROM users WHERE userID='' UNION
SELECT cardNum, sCode, cardExp from creditCards -- AND pass=''
```

Assuming that there is no $userID$ equal to " in the database, the first query (before the UNION term) returns the null set, whereas the second query (after the UNION term) returns data from the $creditCards$ table. In this query, the database would return the attributes, including card number ($cardNum$), security code ($sCode$), expiration date ($cardExp$), which in turn are sensitive information the attacker wants. In our schema reuse system, without linking sensitive attributes back to original schemas, a brute-force attack might be the only way to determine the names of schemas and attributes.