Towards optimal k-anonymization

by Tiancheng Li *, Ninghui Li
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Data & Knowledge Engineering xxx (2007) xxx–xxx
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Towards optimal *k*-anonymization

Tiancheng Li *, Ninghui Li

CERIAS and Department of Computer Science, Purdue University, 305 N. University Street, West Lafayette, IN 47907-2107, USA

Abstract

When releasing microdata for research purposes, one needs to preserve the privacy of respondents while maximizing data utility. An approach that has been studied extensively in recent years is to use anonymization techniques such as generalization and suppression to ensure that the released data table satisfies the *k*-anonymity property. A major thread of research in this area aims at developing more flexible generalization schemes and more efficient searching algorithms to find better anonymizations (i.e., those that have less information loss).

This paper presents three new generalization schemes that are more flexible than existing schemes. This flexibility can lead to better anonymizations. We present a taxonomy of generalization schemes and discuss their relationship. We present enumeration algorithms and pruning techniques for finding optimal generalizations in the new schemes. Through experiments on real census data, we show that more-flexible generalization schemes produce higher-quality anonymizations and the bottom-up works better for small *k* values and small number of quasi-identifier attributes than the top-down approach.

Keywords: Privacy; Anonymization; Generalization

1. Introduction

Organizations, industries and governments are increasingly publishing microdata (i.e., data that contain unaggregated information about individuals) for data mining purposes, e.g., for studying disease outbreaks or economic patterns. While the released datasets provide valuable information to researchers, they also contain sensitive information about individuals whose privacy may be at risk. A major challenge is to limit disclosure risks to an acceptable level while maximizing data utility. To limit disclosure risk, Samarati and Sweeney [17] and Sweeney [18,19] introduced the *k*-anonymity privacy requirement, which requires each record in an anonymized table to be indistinguishable with at least $k-1$ other records within the dataset, with respect to a set of quasi-identifier attributes. To achieve the *k*-anonymity requirement, Samarati and Sweeney [17] and Sweeney [19] used both generalization and suppression for data anonymization. *Generalization* replaces a value with a "less-specific but semantically consistent" value. Tuple suppression removes an entire

[*] Corresponding author. Tel.: +1 765 586 7289.
E-mail address: li83@cs.purdue.edu (T. Li).

0169-023X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2007.06.015

record from the table. Unlike traditional privacy protection techniques such as data swapping and adding noise, information in a *k*-anonymized table through generalization and suppression remains truthful.

A major thread of research in the area of data anonymization aims at generating *k*-anonymous tables with better data quality (i.e., less information loss). This thread of research has resulted in a number of generalization schemes. Each generalization scheme defines a space of valid generalizations. A more flexible scheme allows a larger space of valid generalizations. Given a larger solution space, an optimal generalization in the space is likely to have better data quality. A larger space also requires a better search algorithm.

Samarati and Sweeney [17] and Sweeney [19] used a generalization scheme that utilizes a value generalization hierarchy (VGH) for each attribute. In a VGH, leaf nodes correspond to actual attribute values, and internal nodes represent less-specific values. Fig. 1 shows a VGH for the *work-class* attribute. In the scheme in [17,19], values are generalized to the same level of the hierarchy. One effective search algorithm for this scheme is Incognito, due to LeFevre et al. [9]. Iyengar [8] proposed a more flexible scheme, which also uses a fixed VGH, but allows different values of an attribute to be generalized to different levels. Given the VGH in Fig. 1, one can generalize *Without Pay* and *Never Worked* to *Unemployed* while not generalizing *State-gov*, *Local-gov*, or *Federal-gov*. Iyengar [8] used genetic algorithms to perform a heuristic search in the solution space. Recently, Bayardo and Agrawal [4] introduced a more flexible scheme that does not need a VGH. Instead, a total order is defined over all values of an attribute, and any order-preserving partition (i.e., no two blocks in the partition overlap) is a valid generalization. This scheme has a much larger solution space than previous schemes. Bayardo and Agrawal [4] used the approach in [15] to systematically enumerate all anonymizations and the OPUS framework by Webb [21] to search for the optimal anonymization. They developed several effective pruning techniques to reduce the search space that needs to be explored.

The work in this paper is motivated by three observations. First, the scheme proposed by Bayardo and Agrawal [4] requires a total order on the attribute values. However, it is difficult to define a total order for a categorical attribute and such a total order limits the possible solutions. For example, consider the attribute in Fig. 1, assume that one orders the values from left to right; then generalizations that combine *State-gov* and *Federal-gov* but not *Local-gov* are not considered in this scheme. Second, a VGH reflects valuable information about how one wants the data to be generalized; this is not utilized in [4]. Again consider Fig. 1, it is more desirable to combine *State-gov* with *Local-gov* than with *Private*. Therefore, one may combine *State-gov* with *Private* only when all values under *Government* have been combined together. In other words, one could use VGHs to eliminate some undesirable generalizations. Third, the search algorithm in [4] is a top-down approach, which starts from the most general generalization, and gradually specializes it. Such an approach works well when the value *k* is large. For smaller *k*, a bottom-up search is likely to find the optimal generalization faster.

In this paper, we improve the current state of the art by proposing three new generalization schemes. Given a categorical attribute, these schemes allow any partition of an unordered set of values to be treated as a valid generalization. They also allow a VGH to be used to eliminate some undesirable generalizations. We present a taxonomy of existing generalization schemes and the new schemes proposed in this paper, and analyze the relationship among them.

We also develop an approach for systematically enumerating all partitions in an unordered set. Bayardo and Agrawal [4] used algorithms developed in the artificial intelligence community [15] for enumerating all

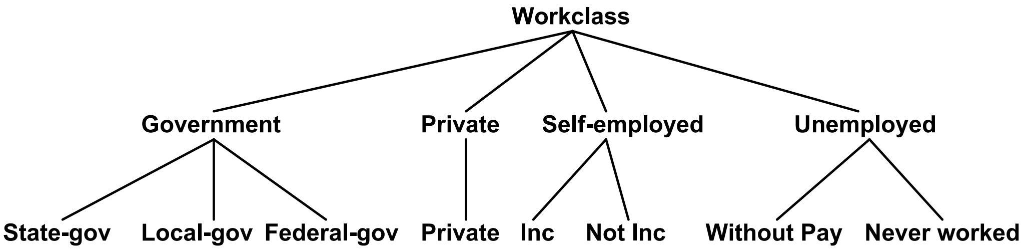


Fig. 1. A value generalization hierarchy for the attribute ***work-class***.

partitions in an ordered set. As we could not find an existing algorithm for the unordered case, we developed an enumeration algorithm. We believe that such an algorithm may be useful in other contexts.

We perform experiments to compare the performance of these generalization schemes and demonstrate that optimal k-anonymizations can be obtained for various generalization schemes and flexible generalization schemes can produce better-quality datasets at the cost of reasonable performance degradation. We find the optimal anonymization in a bottom-up manner. Comparing with the performance of top-down methods in [4], we conclude that bottom-up methods are more suitable for smaller k values while top-down methods are more suitable for larger k values.

The rest of the paper is organized as follows. We present new generalization schemes and a taxonomy of these schemes in Section 2. In Section 3, we give enumeration algorithms for the three new generalization schemes. Cost metrics and pruning rules are discussed in Section 4.4 and experimental results are given in Section 5. We discuss related work in Section 6 and conclude in Section 7.

2. A taxonomy of generalization schemes

In this section, we describe some notations and discuss generalization schemes and their relationship. We also present a taxonomy of these generalization schemes.

2.1. Preliminaries

The attribute domain of an attribute is the set of all values for the attribute. An attribute generalization g for an attribute is a function that maps each value in the attribute domain to some other value. The function g induces a partition among all values in the attribute's domain. Two values v_i and v_j are in the same partition if and only if $g(v_i) = g(v_j)$.

An *anonymization* of a dataset D is a set of *attribute generalizations* $\{g_1, \ldots, g_m\}$ such that there is one attribute generalization for each attribute in the quasi-identifier. A tuple $t = (v_1, \ldots, v_m)$ in D is transformed into a new tuple $t' = (g_1(v_1), \ldots, g_m(v_m))$.

Another anonymization technique is *tuple suppression*, which removes the entire record from the table. Tuple suppression can be very effective when the dataset contains outliers. By removing outliers from the table, much less generalization is needed and the overall data quality improves. Tuple suppression can be incorporated into the framework of generalization by first transforming the dataset D into a new dataset D' using anonymization g and then deleting any tuples in D' that fall into an equivalence class of size less than k. Anonymizations that do not allow suppression can be modeled by assigning the penalty of a suppressed tuple to be infinity. Before discussing algorithms for finding optimal anonymizations, we investigate several generalization schemes.

2.2. Existing generalization schemes

2.2.1. Basic Hierarchical Scheme (BHS)

Earlier work on k-anonymity focuses on the Basic Hierarchical Scheme (BHS), for example [9,17,18]. In BHS, all values are generalized to the same level of the VGH. Thus the number of valid generalizations for an attribute is the height of the VGH for that attribute. For example, there are 3 valid generalizations for the attribute *work-class* in Fig. 1. As BHS has a very small space for valid generalizations, it is likely to suffer from high information loss due to unnecessary generalizations. This motivated the development of other more flexible generalization schemes.

2.2.2. Group Hierarchical Scheme (GHS)

Iyengar [8] proposed a more-flexible Group Hierarchical Scheme (GHS). Unlike BHS, GHS allows different values of one attribute to be generalized to different levels. In GHS, a valid generalization is represented by a "cut" across the VGH, i.e., a set of nodes such that the path from every leaf to the root encounters exactly one node (the value corresponding to the leaf will be generalized to the value in that node). GHS allows a much larger space of valid generalizations than BHS. For example, for the VGH in Fig. 1, there are $2^4 + 1 = 17$

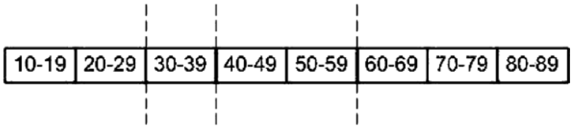


Fig. 2. Partition on continuous attribute ***Age***.

valid generalizations.[1] GHS is likely to produce better-quality anonymizations than BHS. However, the solution space is still limited by the VGH, and the quality of the resulted dataset depends on the choice of the VGH.

2.2.3. Ordered Partitioning Scheme (OPS)

The fact that the quality of the resulted dataset depends on the choice of VGHs motivated the Ordered Partitioning Scheme (OPS) by Bayardo and Agrawal [4]. OPS does not require predefined VGHs. Instead, a total order is defined over each attribute domain. Generalizations are defined by a partition according to the ordering. For example, a partition of the *age* attribute domain is given in Fig. 2.

Suppose an attribute domain contains n values, the total number of generalizations is 2^{n-1}. For example, the total number of valid generalizations for the *work-class* attribute in Fig. 1 is $2^7 = 128$. While the solution space is exponentially large, Bayardo and Agrawal [4] showed the feasibility of finding the optimal solution in OPS through a tree-search strategy exploiting both systematic enumerating and cost-based pruning.

2.3. New generalization schemes

We propose three new generalization schemes, each of which aims at producing higher-quality datasets by allowing a larger solution space while incorporating semantic relationships among values in an attribute domain.

2.3.1. Set Partitioning Scheme (SPS)

OPS requires a pre-defined total order over the attribute domain. While it is natural to define a total order for continuous attributes, defining such a total order for categorical attributes is more difficult. Moreover, this total order unnecessarily imposes constraints on the space of valid generalizations. Consider Fig. 1, suppose the total order is defined using the left-to-right order, then OPS does not allow generalizations that combine *State-gov* and *Federal-gov* but not *Local-gov*; OPS also does not allow generalizations that combine the three values $\{$*Private*, *Without Pay*, *Never Worked*$\}$, without *Inc* and *Not inc*.

We propose the Set Partitioning Scheme (SPS), in which generalizations are defined without the constraint of a predefined total order or a VGH; each partition of the attribute domain represents a generalization. In Section 3, we discuss in detail how to enumerate all valid generalizations in SPS. The number of different partitions of a set with n elements is known as the Bell number Rota [14], named in honor of Eric Temple Bell. They satisfy the recursion formula: $B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}B_k$. The first few Bell numbers are: $B_0 = B_1 = 1$, $B_2 = 2$, $B_3 = 5$, $B_4 = 15$, $B_5 = 52, \ldots$. There are $B_8 = 4140$ generalizations for the *work-class* attribute shown in Fig. 1, as compared to 128 generalizations in OPS. SPS is the most flexible generalization scheme among generalization schemes with the consistency property.

2.3.2. Guided Set Partitioning Scheme (GSPS)

SPS does not take into account the semantic relationship among values of an attribute domain. For example, the three values *State-gov*, *Local-gov*, and *Federal-gov* are semantically related while *State-gov* and *Private* are not.

[1] Except for the most general generalization, each "cut" contains a subset of the four nodes in the middle layer and some nodes on the leaf layer. Thus the total number of "cuts" is one plus the number of subsets of the four nodes in the middle layer.

To incorporate such semantic information, we propose the Guided Set Partitioning Scheme (GSPS), which generalizes data based on the VGHs. GSPS defines a generalization g to be valid if whenever two values from different groups are generalized to the same value v, all values in that two groups should all be generalized to v. If we define the semantic distance between two values to be the height of the lowest common ancestor of the two values in the VGH, then the intuitive idea behind GSPS is that if two values x and y are in one partition, then any value that is semantically closer to x than y must also be in the same partition. (The same applies to any value that is semantically closer to y than x.) Note that a value that has the same semantic distance to x as y does not need to be in the same partition. For example, consider the VGH for *work-class* attribute shown in Fig. 1, if *Local-gov* and *Inc* are combined together, then the five values (*State-gov*, *Local-gov*, *Federal-gov, Inc*, *Not Inc*) must be in the same partition while the other three values do not need to be in that partition.

We can view SPS as a special case of GSPS. GSPS becomes SPS when the VGH is degenerated, i.e., a VGH that has only two levels: one root at the root level and all values at the leaf level. However, with the constraints imposed by VGHs, undesired generalizations that do not maintain semantic relationship among values of an attribute domain can be eliminated while the time needed to find an optimal anonymization reduces as the search space is smaller.

While both GSPS and GHS use VGHs, they are different in a number of ways. GHS requires that values in the same group be generalized to the same level; whereas in GSPS, values in the same group can be generalized to different levels. GSPS allows a larger space of valid generalizations than GHS does. When no VGH is provided (or one uses the degenerated VGH), there are only two valid generalizations in GHS, while the number of valid generalizations in GSPS is maximized to be the same as in SPS.

2.3.3. Guided Ordered Partitioning Scheme (GOPS)

Similar to SPS, OPS does not keep semantic relationship among values in an attribute domain. Consider the *age* attribute, one may consider [20–29] and [30–39] to be two different age groups and two values in the two groups should not be in the same partition unless the two groups are merged in order to achieve a desired level of anonymity. Thus, partitions such as [28–32] are prohibited.

To impose these semantic constraints, we propose the Guided Ordered Partitioning Scheme (GOPS). GOPS defines a generalization g to be valid such that if two values x and y ($x < y$) from two different groups are in the same partition p_g, any value between the least element in x's group and the largest element in y's group must also be in p_g.

The relationship between GOPS and OPS is the same as that between GSPS and SPS. GOPS reduces to OPS when a degenerated VGH is used.

2.4. Putting it all together

Fig. 3 shows a taxonomy of the generalization schemes. We now analyze the relationship among them with regard to the space of valid generalizations. Given two generalization schemes g_1 and g_2, the notation $g_1 \prec g_2$ means that the space of valid generalizations allowed by g_1 is a proper subset of the space of valid generalizations allowed by g_2. We then have the following relationship:

2.4.1. BHS $\prec$ GHS $\prec$ GOPS

It is easy to see that if all values are generalized to the same level, values in the same group are also generalized to the same level. Also, if we define the total order among all values with respect to the hierarchy. For the "workclass" attribute, we can define the total order: *State-gov* $\prec$ *Local-gov* $\prec$ *Federal-gov* $\prec$ *Private* $\prec$ *Inc* $\prec$ *Not Inc* $\prec$ *Without Pay* $\prec$ *Never worked*. Then we can easily see that any valid generalization in GHS is also valid in GOPS.

2.4.2. GOPS $\prec$ OPS $\prec$ SPS

When no hierarchies are defined, GOPS becomes OPS. When no orderings are defined, OPS becomes SPS. Hierarchies and orderings add more constraints to the definition of valid generalizations.

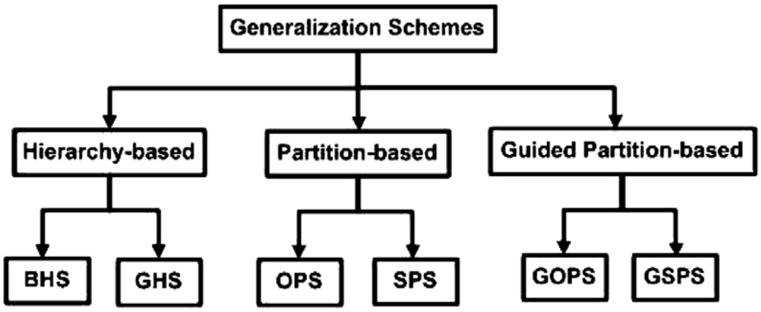


Fig. 3. A taxonomy of generalization schemes.

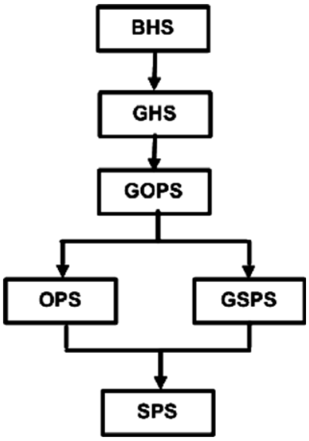


Fig. 4. "Solution space" relationship.

2.4.3. GOPS $\prec$ GSPS $\prec$ SPS

When no orderings are defined, GOPS becomes GSPS. When no hierarchies are defined, GSPS becomes SPS. Hierarchies and orderings add more constraints to the definition of valid generalizations.

The partial order relationship among the six generalization schemes is shown in Fig. 4.

We point out that one can use a combination of generalization schemes for different attributes. For example, one can use SPS for categorical attributes and OPS for continuous attributes.

3. Enumeration algorithms

We now study how to find the optimal anonymizations in the three new generalization schemes: SPS, GSPS and GOPS. To find the optimal anonymization in a scheme, we need to systematically enumerate all anonymizations allowed by the scheme and find the one that has the least cost. The problem of identifying an optimal anonymization in OPS has been framed in [4] as searching through the powerset of the set of all attribute values, which can be solved through the OPUS framework in [21]. OPUS extends a systematic set-enumeration search strategy in [15] with dynamic tree arrangement and cost-based pruning for solving optimization problems. The set-enumeration strategy systematically enumerates all subsets of a given set through tree expansion. See [4] for a description of the algorithm.

In Section 3.1 we present our algorithm for enumerating all generalizations of a single attribute in SPS using tree expansion. In Section 3.2, we present an algorithm for enumerating all anonymizations in SPS,

Please cite this article in press as: T. Li, N. Li, Towards optimal *k*-anonymization, Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.06.015

which revokes the algorithm for a single attribute in Section 3.1. We describe how to adapt the algorithms for GOPS and GSPS in Section 3.3.

3.1. An enumeration algorithm for a single attribute in SPS

Let Σ be the domain of one attribute. In SPS, each generalization for the attribute corresponds to one partition of Σ. A partition of Σ is a family of mutually disjoint sets $S_1, S_2, \ldots, S_m$, such that $\Sigma = S_1 \cup S_2 \cup \cdots \cup S_m$. Our objective is to enumerate all partitions on Σ without visiting any partition more than once. We use breadth-first search (BFS) strategy to build an enumeration tree of all partitions of Σ. The root of the tree is the partition in which each value itself is in a set; this represents the most specific generalization, where no value is generalized. Each child of the node is generated by merging two sets in the partition into one set. The challenge is to generate each partition exactly once. Before describing the algorithm, we show the partition enumeration tree for the alphabet $\{1,2,3,4\}$ in Fig. 5. This may help understand the key ideas underlying the enumeration algorithm.

Given a node that has partition $P = \langle S_1, \ldots, S_t \rangle$, a child node of P is generated by merging two sets S_j and S_i ($1 \leqslant j < i \leqslant t$) in P. If all pairs of S_i and S_j are allowed to be merged, then a partition may be encountered multiple times. The challenge is to identify under which conditions can S_i and S_j merge so that each partition is generated exactly once.

The algorithm is given in Fig. 6. The key component of the algorithm is the Child_Nodes procedure which finds all the child nodes of a given partition P. In the algorithm, two sets S_j and S_i can be merged if and only if *all three* of the following conditions are satisfied. For each of the condition, we briefly explain the intuition behind it.

(1) S_i contains a single element e. Suppose that $S_i = \{e_1, e_2\}$, then the child partition with S_i and S_j merged can be generated elsewhere in the tree with S_j first merged with $\{e_1\}$ and then with $\{e_2\}$.
(2) *Each set in between (i.e., $S_{j+1}, \ldots, S_{i-1}$) contains a single element.* Suppose there is a k such that $j+1 \leqslant k \leqslant i-1$ and S_k contains more than one element, then elsewhere in the tree, we have a partition in which S_k is replaced by several sets, each of which contains exactly one element. The partition with S_i and S_j merged will be generated there.
(3) *Each element in S_j is less than e.* Suppose that $S_j = \{e_1, e_2\}$ with $e_1 < e < e_2$, then the partition with S_i and S_j merged can be generated elsewhere in the tree with $\{e_1\}$ first merged with $\{e\}$ and then with $\{e_2\}$. Note that S_j must contain an element that is less than e, because S_j comes before S_i.

For each set S_i in P, the algorithm checks if S_i contains more than one element. If so, S_i cannot be merged with any set preceding it. Otherwise (S_i contains exactly one element e), the algorithm checks preceding sets S_j of S_i, starting from S_{i-1}. If every element in S_j is less than e, a new child partition is generated by removing S_i and S_j, and adding $S_i \cup S_j$ as the jth element of the partition. Otherwise (some element in S_j is larger than e), S_i

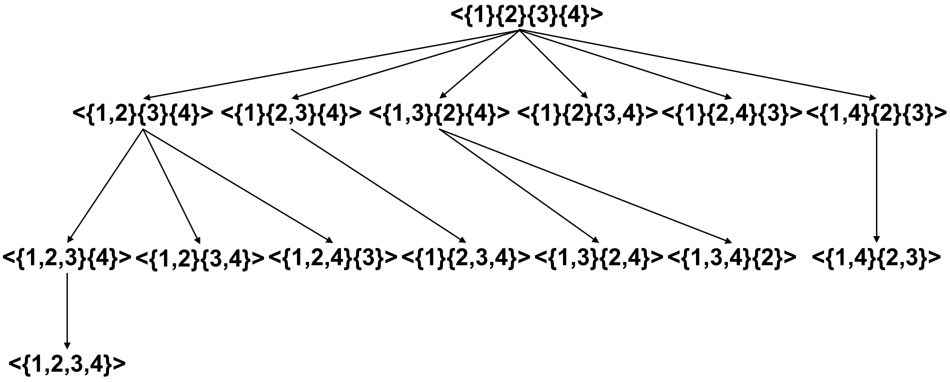


Fig. 5. Partition enumeration tree over alphabet $\{1,2,3,4\}$.

```
Enumeration_Algorithm
Input: An attribute domain S
Output: The set of all partitions on S
  Initialize the partition P=$\langle S_1, S_2, ..S_m\rangle$
      s.t. each set contains one element.
  Insert P into queue
  $R = \emptyset$
  while the queue is not empty do
      P = first item removed from queue
      T = Child_Partitions(P)
      for each partition $P'$ in T do
          Insert $P'$ to queue
      $R = R \cup P$
  return R
```

```
Child_Partitions
Input: A partition P=$\langle S_1, ..., S_t\rangle$ on S
Output: The set of child partitions of P
  $T = \emptyset$
  for i=2 to t do
      if $S_i$ contains one element e then
          for j=i-1 to 1 do
              if e > any element in $S_j$ then
                  $T = T \cup \{\langle S_1, S_2...S_{j-1}, S_j \cup S_i, S_{j+1}, ...S_{i-1}, S_{i+1}, ...S_t\rangle\}$
              else break
              if $S_j$ has two or more elements
              then break
  return T
```

Fig. 6. Enumeration algorithm for a single attribute.

cannot be merged with any set preceding S_j. If S_j contains more than one element, S_i cannot be merged with any set preceding S_j either.

Example 1. Consider the partition $\langle\{1\},\{2,3\},\{4\},\{5\}\rangle$. This partition has three child partitions by merging {4} with {2,3}, or merging {5} with {4}, or merging {5} with {2,3}. The resulting partitions are $\langle\{1\},\{2,3,4\},\{5\}\rangle$, $\langle\{1\},\{2,3\},\{4,5\}\rangle$ and $\langle\{1\},\{2,3,5\},\{4\}\rangle$.

Example 2. Consider the partition $\langle\{1,4\},\{2\},\{3\},\{5\}\rangle$. This partition has four children by merging {3} with {2}, or merging {5} with {3}, or merging {5} with {2}, or merging {5} with {1,4}. The resulted partitions are $\langle\{1,4\},\{2,3\},\{5\}\rangle$, $\langle\{1,4\},\{2\},\{3,5\}\rangle$, $\langle\{1,4\},\{2,5\},\{3\}\rangle$, and $\langle\{1,4,5\},\{2\},\{3\}\rangle$.

The following theorem states the correctness of the algorithm.

Theorem 1. *The algorithm in Fig. 6 enumerates all partitions of S in a systematic manner, i.e., each partition of S is enumerated exactly once.*

Proof Sketch. Consider a partition $P = \langle\{a_{11}, a_{12} \ldots a_{1t_1}\}, \{a_{21}, a_{22}, \ldots a_{2t_2}\}, \ldots, \{a_{s1}, a_{s2}, \ldots a_{st_s}\}\}\rangle$ of S, such that (1) $a_{ij} < a_{ik}$ for $i = 1, 2, \ldots, s$ and $1 \leqslant j < k \leqslant t_i$. (2) $a_{j1} < a_{k1}$ for $1 \leqslant j < k \leqslant s$. We show that there is exactly one valid sequence of merging that results in this partition; this shows that the partition is generated exactly once in the tree.

In order to make the proof concise, we denote "merging e to the set s" as $\langle e, s\rangle$. Then we give an order of merging that results in P from the initial partition P_o:$\langle a_{12}, \{a_{11}\}\rangle, \langle a_{13}, \{a_{11}, a_{12}\}\rangle, \ldots, \langle a_{1t_1}, \{a_{11}, a_{12}, \ldots a_{1t_1-1}\}\rangle$, $\langle a_{22}, \{a_{21}\}\rangle, \langle a_{23}, \{a_{21}, a_{22}\}\rangle, \ldots, \langle a_{2t_2}, \{a_{21}, a_{22}, \ldots a_{2t_2-1}\}\rangle, \ldots, \langle a_{s2}, \{a_{s1}\}\rangle, \langle a_{s3}, \{a_{s1}, a_{s2}\}\rangle, \ldots, \langle a_{st_s}, \{a_{s1}, a_{s2}, \ldots a_{st_s-1}\}\rangle$.

We can easily see that all $\sum_{i=1}^{s}(t_i - 1)$ merges are valid and therefore the partition P is enumerated in our algorithm.

We can show that the above ordering is unique through two observations:

(1) a_{ij} must be merged before a_{ik} for any $i = 1, 2, \ldots, s$ and $1 \leqslant j < k \leqslant t_i$. Since $a_{ij} < a_{ik}$ and a_{ij} cannot be merged with a set that contains a_{ik} which is a larger element than a_{ij}.

(2) a_{ip} must be merged before a_{jq} for any $1 \leqslant i < j \leqslant s$, $1 < p \leqslant s_i$ and $1 < q \leqslant s_j$. Two cases are identified:
   - $a_{ip} < a_{jq}$. Since if an element is merged, any other elements before it cannot be merged, we see that a_{ip} must be merged first.
   - $a_{ip} > a_{jq}$. Since an element cannot be merged with any set before a set which contains more than one element, a_{ip} must be merged earlier than a_{jq}. □

Please cite this article in press as: T. Li, N. Li, Towards optimal *k*-anonymization, Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.06.015

We have shown that our algorithm enumerates all partitions on S and each partition is enumerated exactly once. The enumeration algorithm is thus systematic. As we have mentioned, the total number of valid generalizations with n values in SPS is Bell Number $B(n)$. The complexity of the enumeration algorithm is thus $O(B(n))$. The asymptotic formula of $B(n)$ (according to wikipedia) is $B(n) \sim \frac{1}{\sqrt{n}}[\lambda(n)]^{n+\frac{1}{2}}e^{\lambda(n)-n-1}$ where $\lambda(n) = e^{W(n)}$ and $W(n)$ is the Lambert W function and satisfies $W(n)e^{W(n)} = n$.

Note that our enumeration algorithm is "bottom-up" in that it starts from the original dataset and incrementally generalizes the dataset until every value is generalized to the most general value. However, in the representation of the enumeration tree, the top node represents the original dataset and a child node represents a more general generalization. Although our algorithm visits all nodes in a "top-down" manner, our algorithm is a "bottom-up" algorithm.

3.2. An anonymization enumeration algorithm for SPS

Recall that an anonymization is a set of attribute generalizations $\{P_1, P_2, \ldots, P_m\}$ consisting of one attribute generalization per attribute. In this section, we build an enumeration tree to enumerate all possible anonymizations. Each node in the enumeration tree has m attribute generalizations (one for each attribute) and an *applicator* set. An *applicator* set is an ordered subset of $\{1, \ldots, m\}$, denoting the order in which the attributes are to be expanded. By applying each applicator in the applicator set of a node, we obtain a set of children of that node. For example, the first set of children of a node is the set of anonymizations created by generalizing the attribute specified by the first applicator. A child of a node inherits all other applicators and inherits the applicator that has been applied if the attribute corresponding to the applicator can still be generalized. Fig. 8 shows an enumeration tree of two attributes with three and two values, respectively.

Fig. 7 shows an algorithm using Breadth-First Search (BFS) strategy to systematically enumerate all possible anonymizations. The *Anonymization_Enumeration* procedure uses a queue structure. Each time a node is removed from the queue, all its children computed by the *Child_Nodes* procedure are inserted to the queue. The *Child_Nodes* procedure applies each applicator in the applicator set to the anonymization and calls the *Child_Partitions* procedure in Fig. 6 to find all child partitions of the given partition. This child partition replaces the original partition in the anonymization and the applicator set is updated according to whether the child partition can still be generalized or not.

Example 3. Consider a node $\{\langle\{1,2\},\{3\},\{4\}\rangle, \langle\{1\},\{2\}\rangle\}$ with $AS = \{1,2\}$. By applying the first applicator 1, we obtain three child nodes, namely, $\{\langle\{1,2,3\},\{4\}\rangle,\langle\{1\},\{2\}\rangle\}$, $\{\langle\{1,2\},\{3,4\}\rangle,\langle\{1\},\{2\}\rangle\}$, and $\{\langle\{1,2,4\},\{3\}\rangle,\langle\{1\},\{2\}\rangle\}$. By applying the second applicator 2, we obtain one child nodes, namely, $\{\langle\{1,2\},\{3\},\{4\}\rangle,\langle\{1,2\}\rangle\}$. Therefore, this node has four child nodes in total.

```
Anonymization_Enumeration
Input: A set of attribute domains
Output: A set of all anonymizations
  Initialize the node of to the most specific
    partition, and AS contains all attributes
  Insert the initial node to queue
  $R = \emptyset$
  while the queue is not empty do
    N = first item removed from queue
    T = Child_Nodes(N)
    for each node $N'$ in T do
      Insert N' to queue
    $R = R \cup N$
  return R
```

```
Child_Nodes
Input: A tree node $N$(an anonymization
    $\langle P_1, P_2, ...P_m \rangle$ and an applicator set $AS$)
Output: All child nodes of node $N$
  $T = \emptyset$
  for each applicator $i$ in $AS$ do
    parSet=Child_Partitions($P_i$)
    for each partition $P$ in parSet do
      Create a new child node $N'$
      Set anonymization of $N'$ to be
          $\langle P_1, ...P_{i-1}, P, P_{i+1}, ...P_m \rangle$
      Set the applicator set of $N'$ to be $AS$
      if $P$ is the most generalized partition
       then remove $i$ from AS of $N'$
      $T = T \cup N'$
  return T
```

Fig. 7. Enumeration algorithm for anonymizations.

Please cite this article in press as: T. Li, N. Li, Towards optimal *k*-anonymization, Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.06.015

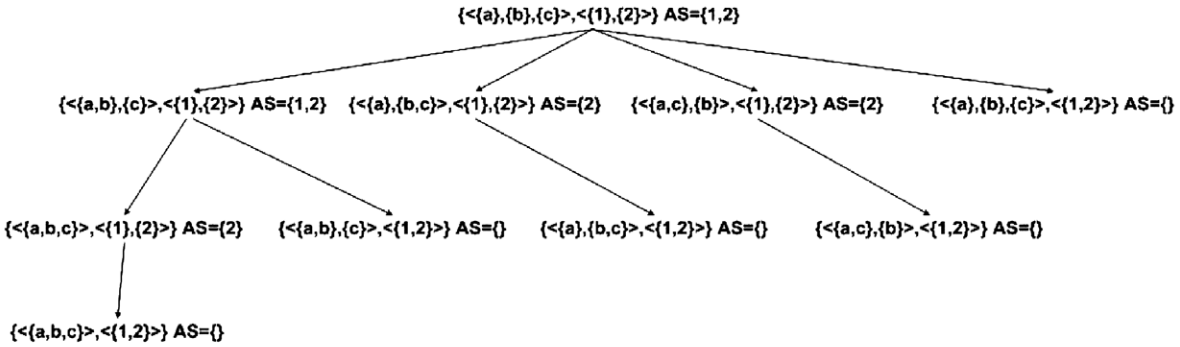

{<{a},{b},{c}>,<{1},{2}>} AS={1,2}
{<{a,b},{c}>,<{1},{2}>} AS={1,2}
{<{a},{b,c}>,<{1},{2}>} AS={2}
{<{a,c},{b}>,<{1},{2}>} AS={2}
{<{a},{b},{c}>,<{1,2}>} AS={}
{<{a,b,c}>,<{1},{2}>} AS={2}
{<{a,b},{c}>,<{1,2}>} AS={}
{<{a},{b,c}>,<{1,2}>} AS={}
{<{a,c},{b}>,<{1,2}>} AS={}
{<{a,b,c}>,<{1,2}>} AS={}

Fig. 8. Enumeration tree of anonymizations with two attributes.

3.3. *Enumeration algorithms for GSPS and GOPS*

The enumeration algorithm for SPS described in Sections 3.1 and 3.2 can be adapted for GSPS. The only difference is that when we expand a node, we examine each of its child nodes to see if this child node represents a valid generalization with respect to the VGH or not. If yes, the child node is added to the queue. Otherwise, the algorithm identifies all sets of attribute values that need to be merged to get a valid generalization and check whether such merging is allowed according to the three conditions described in Section 3.1. If such merging is allowed, then a new node is created. This enumeration approach remains systematic and complete. GSPS allows fewer valid generalizations than SPS since undesired generalizations that violate the VGHs are regarded as invalid generalizations in GSPS. GSPS becomes SPS when degenerated VGHs are used. Therefore, GSPS is a more sophisticated scheme than SPS.

Example 4. Consider the work-class hierarchy in Fig. 1 and the partition $\langle\{1\},\{2,3\},\{4\},\{5,6\},\{7\},\{8\}\rangle$. In SPS, this partition has 4 child partitions. But in GSPS, this partition has only 1 child partition by merging $\{8\}$ with $\{7\}$. The resulted partition is: $\langle\{1\},\{2,3\},\{4\},\{5,6\},\{7,8\}\rangle$. The other 3 child partitions are invalid with regard to the hierarchy.

Enumeration algorithm for OPS can also be adapted for GOPS using the same approach.

Example 5. Consider the work-class hierarchy in Fig. 1 and the partition $\langle\{1\},\{2,3\},\{4\},\{5,6\},\{7\},\{8\}\rangle$. In OPS, this partition has 2 child partitions. But in GOPS, this partition has only 1 child partition by merging $\{8\}$ with $\{7\}$. The resulted partition is: $\langle\{1\},\{2,3\},\{4\},\{5,6\},\{7,8\}\rangle$.

4. Cost metrics and cost-based pruning

In this section, we discuss several cost metrics and compare them in terms of effectiveness in measuring information loss. We then employ cost-based pruning rules to reduce the search space.

4.1. *Cost metrics*

To model an optimal anonymization, we need a cost metric to measure the data quality of the resulted dataset. One widely used metric is the discernibility metric (DM) in [4], which assigns a penalty to each tuple according to the size of the equivalence class that it belongs to. If the size of the equivalence class E is no less than k, then each tuple in E gets a penalty of $|E|$ (the number of tuples in E). Otherwise each tuple is assigned a penalty of $|D|$ (the total number of tuples in the dataset). In other words,

$$C_{\mathrm{DM}} = \sum_{\forall E \text{ s.t. } |E| \geqslant k} |E|^2 + \sum_{\forall E \text{ s.t. } |E| < k} |E||D|$$

DM measures the discernibility of a record as a whole. We propose *Hierarchical Discernibility Metric* (*HDM*), which captures the notion of discernibility among attribute values. For example, consider the *work-class* attribute in Fig. 1, suppose 50 records have value *Inc* and 200 records have value *Not-inc*. If values *Inc* and *Not-inc* are combined (e.g., generalized to *Self-employed*), we would expect a larger information loss for value *Inc* than for value *Not-Inc*.

Given an attribute generalization g and its corresponding partition P, suppose that a record has value v for this attribute, and v is in the group $e \in P$. We quantify the information loss for generalizing v in this record. Let N be the total number of records. Let N_e be the number of records that have values in the group e. Let N_v be the number of records that have value v. In our metric, generalizing values from v to e leads to a penalty of $(N_e - N_v)/(N - N_v)$. For the earlier example, suppose the total number of records is 1000, generalizing *Inc* to *Self-employed* gets a penalty of $(250 - 50)/(1000 - 50) = 4/19$ while the penalty is $(250 - 200)/(1000 - 200) = 1/16$ when *Not-inc* is generalized to *Self-employed*.

The penalty for a single attribute is between 0 and 1. No penalty is incurred when the value is not generalized and a penalty of 1 is incurred when the value is generalized to the most general value. The penalty for a record is the average penalty for each attribute. Therefore, it is also between 0 and 1. Compared with the entropy-based information loss measure proposed by Domingo-Ferrer and Torra [5], our HDM measure is a generalization of the discernibility metric (DM) and can be efficiently computed.

4.2. A comparison of cost metrics

Before we present cost-based pruning techniques, we give a brief comparison of DM and HDM. First and foremost, they differ in that DM calculates discernibility at tuple level, whereas HDM calculates discernibility at cell level. To more clearly understand their similarities and differences, we consider their effect when the data has only one attribute in the quasi-identifier.

When we generalize two values v_A and v_B to a more general value v_C, both metrics assign a larger penalty for the value where fewer records have that value. Suppose that there are n_A records with value v_A and n_B records with value v_B, and we generalize v_A and v_B to v_C where there are $n_C = n_A + n_B$ records having value v_C. Using DM, the extra penalty for records with v_A is $n_C - n_A$ while the extra penalty for records with v_B is $n_C - n_B$. If $n_A > n_B$, then records with v_B will get a larger penalty than those with v_A. The same is true for HDM, where the extra penalty for records with v_A is $(n_C - n_A)/(n - n_A) = 1 - (n - n_C)/(n - n_A)$ and the extra penalty for v_B is $(n_C - n_B)/(n - n_B) = 1 - (n - n_C)/(n - n_B)$. Here, n is the total number of records in the table. If $n_A > n_B$, then v_B will get a larger penalty than v_A. In this aspect, the two metrics are consistent with each other.

The two metrics differ in that HDM considers the relative frequency of a value in the overall table while DM relies only on the relative frequency of a value in the group. In other words, HDM considers the total number of records in the whole table in assigning a penalty to a value while DM does not. Recall that the average penalty for generalizing v_A to v_C in DM is $n_C - n_A$. Therefore, for DM, generalizing a value where 2 records have that value to a group of 4 records is exactly the same for generalizing a value where 1000 records have that value to a group of 1002 records. However, intuitively, the first value should get a larger penalty. Our HDM metric captures that aspect.

Another difference between DM and HDM is that DM is defined on one table whereas HDM is defined based on one generalization. We can also define HDM based on one table as follows. Suppose there are n_A records with the original value v_A, and in one table T_1, v_A is generalized to v_{A1} where there are n_{A1} records with value v_{A1}. Then, the cost associated with table T_1 on value v_{A1} is defined as $n_{A1}/(n - n_A)$. Generalizing v_A to v_{A1} will then take cost $(n_{A1} - n_A)/(n - n_A)$, which is exactly what we have defined for HDM. Generalizing v_{A1} to v_{A2} will take cost $(n_{A2} - n_{A1})/(n - n_A)$. The sum of the two costs is $(n_{A2} - n_A)/(n - n_A)$, which is exactly the cost for generalizing v_A directly to v_{A2}. This shows that our HDM metric satisfies the addition property.

4.3. Cost-based pruning

Using the cost metrics, we can compare the data quality of a dataset produced by an anonymization. The optimal anonymization is defined as one that results in the least cost. To find the optimal anonymization, the naive way may traverse the whole enumeration tree using some standard strategies such as *DFS* or *BFS*. But such an algorithm is impractical when the number of possible anonymizations becomes exponentially large. Some pruning heuristics must be applied in order to reduce the search space and make the algorithm practical. Significant performance improvement can be achieved if we can effectively prune parts of the enumeration tree that will not produce an optimal solution.

In [4], the authors identified a number of pruning rules using a branch and bound approach. Their pruning algorithm first tries to prune the node itself. A node can be pruned only when we are assured that none of its descendants could be optimal. This decision can be made by the lower-bound cost computation, which calculates the lowest cost possible for any node in the subtree rooted at that node. When a node is encountered, the lowest cost for the subtree rooted at that node is computed and compared with the current best cost. If it is no less than the current best cost, the whole subtree rooted at that node can be pruned. If the node cannot be pruned, the algorithm employees useless value pruning which tries to prune value from the applicator set which cannot lead to a better anonymization.

In our bottom-up approach, these two pruning rules can be applied. Starting from the original data, we use BFS to go through the anonymization enumeration tree built in the previous section. We keep track of the current best cost and compare with the lower-bound cost of each node we encounter to decide whether the node can be pruned or not. If not, we compare the lower-bound cost of a new node by applying one of the applicators to decide whether the applicator can be pruned from the applicator set or not.

The key component of the pruning framework is the lower-bound cost computation, which calculates the lowest cost possible for any node in a subtree. In this section, we first describe how to estimate the lower-bound cost that nodes in a subtree can have. Then we discuss several new pruning techniques that can be used to dramatically cut down the search space.

4.3.1. Lower-bound cost computation for HDM

The lower-bound cost of a node N is an estimate of the lowest cost possible for any node in the subtree rooted at N. The lower-bound cost can be used to decide whether a whole subtree can be pruned, i.e., if the lower-bound cost of N is no less than the current best cost, then the whole subtree rooted at N can be pruned.

Calculating the lower-bound cost for DM is described in [4]. We now describe how to calculate lower-bound cost for HDM. Let A be an ancestor of node N. We denote the penalty assigned to record r at node N as $penalty(N,r)$. Let r_1 be a record that is not suppressed by A. We observe that r_1 is also not suppressed by N. Moreover, the equivalence class that contains r_1 at A is a subset of the equivalence class that contains r_1 at N and therefore $penalty(N,r_1) \geqslant penalty(A,r_1)$. Let r_2 be a record that is suppressed by A. Then r_2 may be suppressed by N or not. $penalty(N,r_2)$ can be as small as *0*.

Based on the above argument, we can compute the low-bound cost of node A as $LB_{\mathrm{HDM}}(A)$

$$LB_{\mathrm{HDM}}(A) = \sum_{r \in D} \begin{cases} penalty(A,r) & \text{if } r \text{ is not suppressed} \\ 0 & \text{otherwise} \end{cases}$$

Since the applicability of pruning rules is dependent on what cost metric is used. Here, we identify the properties that a cost metric should have so that the pruning rules are applicable:

(1) Penalty for a suppressed record should be at least as high as that for an unsuppressed record.
(2) If an unsuppressed record is generalized, the penalty for that record increases after the generalization.

This two requirements on cost metric are both sufficient and necessary for the pruning rules to be applicable. Below we identify two kinds of pruning rules: node pruning and applicator pruning.

4.4. New pruning techniques

In this paper, we introduce a new type of pruning technique: useful applicator pruning. This category of rules tries to identify applicators that must be applied in order to reach an optimal solution and prune nodes that do not generalize on that applicator. Such an applicator is called useful. Then we can prune nodes that do not include that applicator. The following criteria identifies useful applicators.

Useful applicators can be identified by checking whether the applicator is the only one that can lead to a k-anonymized table. Specifically, if for any combination of applicators other than v, there exists a record r such that r falls into an equivalence class of size less than k, then v is a useful applicator since only by generalizing on v we can have a k-anonymized table without suppressing record r. However, this has the limitation that we require all records satisfy k-anonymity property and suppression is not allowed.

For our pruning techniques to be effective, it is imperative that we find an anonymization close to the optimal anonymization early, since it can then be used to eliminate a large number of suboptimal anonymizations. We propose two techniques that can be used effectively to identify an anonymization that is close to the optimal anonymization, i.e., find a cost that is close to the best cost:

4.4.1. Seeding

Seeding involves the initialization of the best cost. The initial best cost can be set as the cost associated with the original dataset(e.g., $|D| * |D|$ for DM and $|D|$ for HDM). However, more pruning can be done if the initial best cost value can be estimated more precisely. For example, the initial best cost can be estimated using costs associated with a set of randomly selected nodes.

4.4.2. Modified BFS search strategy

We modify the simple BFS search strategy to achieve this. One solution is that when we find a node whose lower-bound cost is smaller than the current best cost, we do not immediately add all its children to the queue. Instead, we add that node to the queue for later re-consideration. Since the cost associated with that node has already been computed, it is available when it is retrieved from the queue for the second time. At that point, since the current best cost may have decreased, it is likely that the lower-bound cost of that node is larger than the current best cost, in which case the whole subtree rooted at that node can be pruned.

During our search process, we often need to select a node from the queue or an applicator from the applicator set as the next node or applicator for consideration. The choice of a good node or application selection order would eliminate a large number of nodes or application from examination.

4.4.3. Node rearrangement

At each step of the search algorithm, we choose one node from the queue for consideration. In simple BFS, we choose the node at the front of the queue to be the next node for consideration. A better approach is to choose the node with smallest lower-bound cost, with the hope that the best cost can be identified more quickly.

4.4.4. Applicator rearrangement

Once we decide to consider a node, we need to apply one applicator to get its children. But which applicator to use is a subjective issue. One approach is to order all the applicators according to ascending order of how many equivalence classes are merged by generalizing on that applicator. A good choice of the next applicator to be applied can improve the performance of the algorithm; otherwise, good anonymizations are distributed uniformly among the search tree.

We will evaluate and compare the effectiveness of different pruning techniques in cutting down the search space in the experiment.

5. Experiments

The goal of the experiments is to compare the performance (both in terms of efficiency and data quality) of different generalization schemes, the efficiency of the bottom-up approach and the top-down approach, and the effectiveness of different pruning techniques. To achieve this goal, we implemented all six generalization schemes and performed experiments using a real-world dataset.

5.1. Experimental setup

The dataset used in the experiments is the adult dataset from the UC Irvine machine learning repository, which is comprised of data collected from the US census. We used nine attributes of the dataset, as shown in the following figure. Records with missing values are eliminated and there are 30162 valid records in total. The dataset used in the experiment is described in Fig. 9. The algorithms are implemented in JAVA and experiments are run on a 3.4 GHz Pentium 4 machine with 2 GB Physical Memory Space.

5.2. *Experimental results*

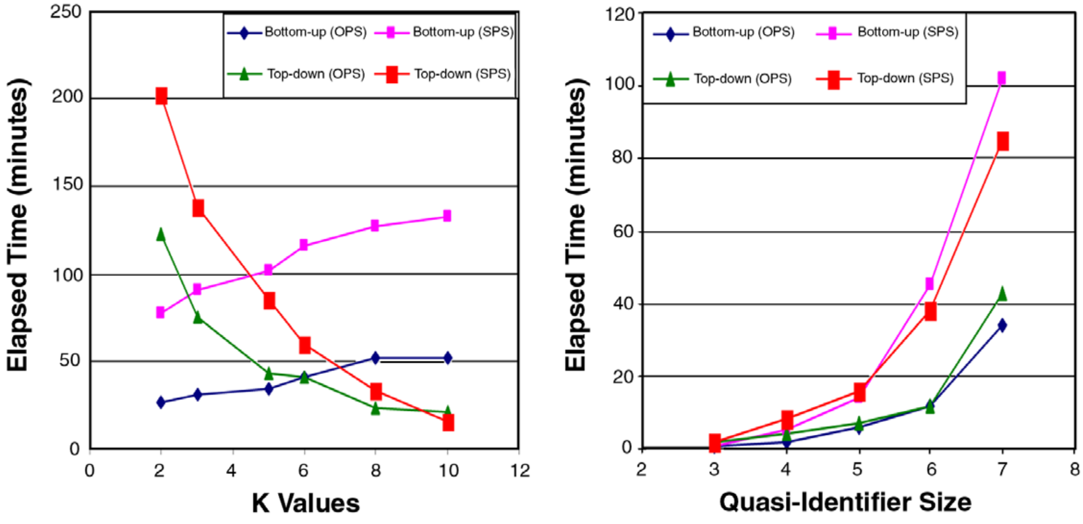
We use coarse partitioning on the *age* attribute, where the domain was pre-partitioned into 15 intervals, with each interval containing exactly a 5-year range. Using coarse partitioning, the search space is reduced dramatically while still large enough to define the optimal anonymization.

5.2.1. *Efficiency comparisons of the bottom-up approach and the top-down approach*

Our first experiment compares the efficiency of the bottom-up approach with that of the top-down approach. We first compare the two approaches using fixed four QI values: $\{Age, Marital\_Status, Race, Gender\}$. Fig. 10a shows the Efficiency of the bottom-up approach and the top-down approach with varied k values using OPS and SPS. As we can see, the bottom-up approach runs faster than the top-down approach for small k values like 2 or 3. However, for larger k values like 10, 15 or 20, the top-down approach finds the optimal anonymization faster. This is because for smaller k values, the original dataset does not need to be gen-

	Attribute	Type	# of values	Height
1	Age	Numeric	74	5
2	Work-class	Categorical	8	3
3	Education	Categorical	16	4
4	Country	Categorical	41	3
5	Marital–Status	Categorical	7	3
6	Race	Categorical	5	3
7	Gender	Categorical	2	2
8	Occupation	Sensitive	14	3
9	Salary	Sensitive	2	2

Fig. 9. Description of the **Adult** dataset used in the experiment.



(a) Efficiency (with varied k values)

(b) Efficiency (with varied QI size)

Fig. 10. Efficiency comparisons of the bottom-up approach and the top-down approach.

Please cite this article in press as: T. Li, N. Li, Towards optimal k-anonymization, Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.06.015

eralized much in order to achieve k-anonymity. Therefore, a bottom-up approach which starts from the original dataset would find the optimal anonymization faster. On the contrary, for larger k values, a top-down approach would run faster since the dataset has to be generalized much to achieve k-anonymity.

We also compare the two approaches using varied QI size. Fig. 10b shows the performances of the two approaches with regard to different QI size using OPS and SPS. From the figure, we see that the bottom-up approach outperforms the top-down approach when QI size is small and the top-down approach works better when the QI size is large. For smaller QI size, few generalization steps are needed in order to achieve k-anonymity. Therefore, the bottom-up approach would find the optimal anonymization faster. On the contrary, when the QI size is large, most of the attributes have to be generalized to high levels on the taxonomy tree. This is consistent with the finding by Aggarwal [1] that large amount of information has to be lost in order to achieve k-anonymity, especially when the data contains a large number of attributes.

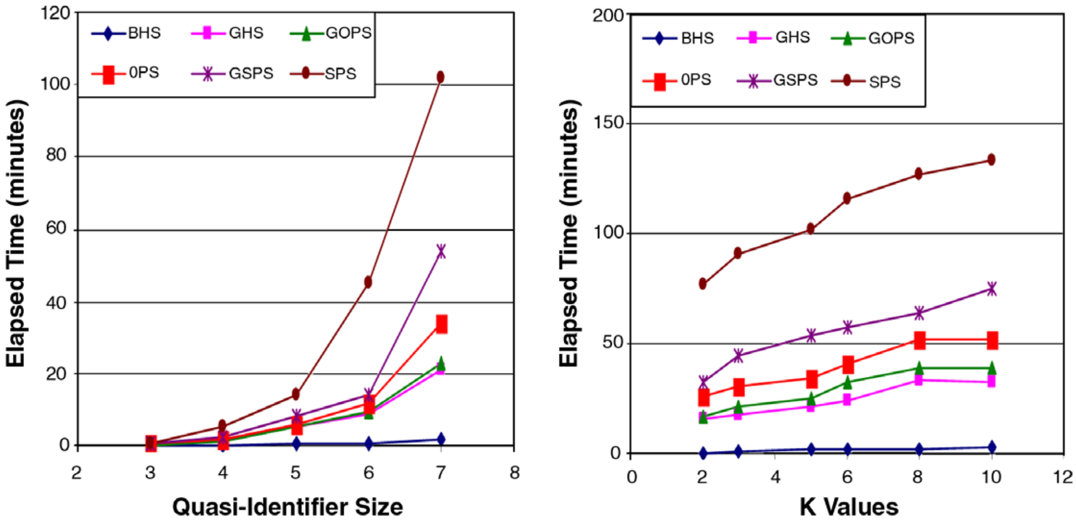
5.2.2. Efficiency comparisons of different generalization schemes

Our second experiment compares the efficiency of various generalization schemes. We first compare the efficiency with varied quasi-identifier size, shown in Fig. 11a with fixed $k = 5$. As we expect, the exponentially increasing search space greatly increases the running time. Also, for the same generalization scheme, the running time increases as we use a larger quasi-identifier.

We also compare the efficiency of the six generalization schemes with varied k values. Fig. 11b shows the experimental results. Since we use a bottom-up search method, we would expect to find the optimal solution very quickly for small k values. As we expect, the running time of the generalization schemes increases as k increases for each generalization scheme. The data reported in [4] shows that a top-down search method can find the optimal solution quickly for larger k values. The two search directions thus complement each other.

5.2.3. Data quality comparisons of different generalization schemes

Our third set of experiment measures the data quality of the resulted dataset produced by the six generalization schemes, with varied k values. We measure the data quality by computing the cost associated with the anonymized dataset. The cost metrics used here are DM and HDM discussed in Section 4.1. For the same generalization scheme, the cost increases as k increases. This can be explained by the fact that a larger k value implies higher privacy level, which in turn results in a larger cost. For the same k value, the cost decreases for the more sophisticated generalization schemes. This can be explained by the fact that the more sophisticated generalization schemes allow more valid generalizations and produce a dataset with better data quality.



(a) Efficiency ($k = 5$) (b) Efficiency ($QI = 7$)

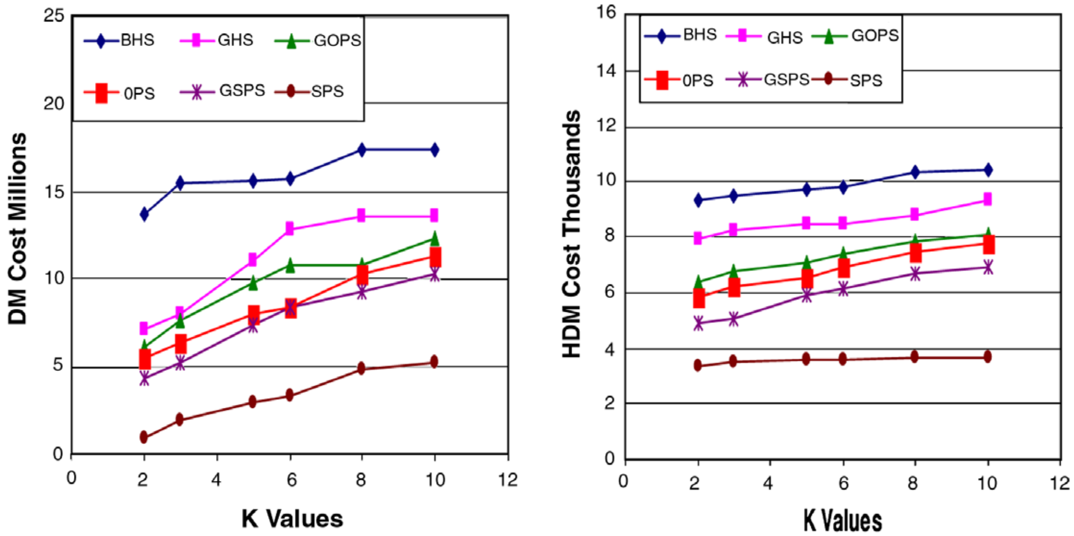
Fig. 11. Efficiency comparisons of the six generalization schemes.

Please cite this article in press as: T. Li, N. Li, Towards optimal k-anonymization, Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.06.015

The experiment results are consistent with our analysis. Fig. 12a shows the discernibility metric cost for the six generalization schemes with varied *k* values. Fig. 12b shows the hierarchical discernibility metric cost for the six generalization schemes with varied *k* values.

5.2.4. Effectiveness of different pruning techniques

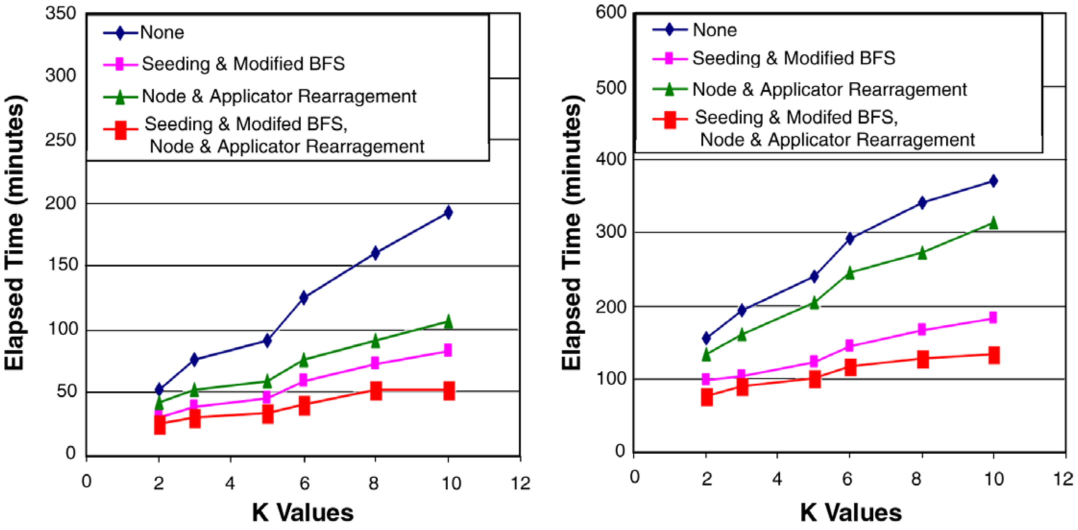
Finally, we experimented with the effectiveness of different pruning techniques in cutting down the search space. We test the two classes of techniques described in Section 4.4: (1) seeding & modified BFS, and (2) node & applicator rearrangement. The results are shown in Fig. 13. As we can see, the use of these two classes of techniques can effectively the performance in finding the optimal anonymizations. The combination of these



(a) Data Quality (DM)

(b) Data Quality (HDM)

Fig. 12. Data quality comparisons of the six generalization schemes.



(a) Using OPS

(b) Using SPS

Fig. 13. Effectiveness comparisons of different pruning techniques.

two techniques could reduce the running time by up to 60%. In general, the first technique is more effective (it can reduce the running time by up to 40%). Thus, early identification of an anonymization close to the optimal anonymization is an effective approach to elimination the examination of a large number of suboptimal anonymizations.

6. Related work

Many generalization schemes have been proposed in the literature to achieve k-anonymity. Most of these schemes require predefined value generalization hierarchies, for example [7–9,16,17,20]. Among these schemes, some require values be generalized to the same level of the hierarchy in [9,16,17]. Iyengar [8] extends previous schemes by allowing more flexible generalizations. In addition to these hierarchy-based schemes, partition-based schemes have been proposed for totally-ordered domains in [4]. These schemes and their relationship with our proposed schemes are discussed in detail in Section 2.

All schemes discussed above satisfy the ‘‘consistency property’’, i.e., multiple occurrences of the same attribute value in a table are generalized in the same way. There are also generalization schemes that do not have the consistency property. In these schemes, the same attribute value in different records may be generalized to different values. For example, LeFevre et al. [10] propose Mondrian multidimensional k-anonymity, where each record is viewed as a point in a multidimensional space and an anonymization is viewed as a partitioning of the space into several regions. Another technique to achieve k-anonymity requirement is clustering, e.g., [6,3]. In this paper, we focus on generalization schemes that have the consistency property. We feel that the consistency property is a desirable property for many usages of the data, especially for data mining applications.

On the theoretical side, optimal k-anonymity has been proved to be NP-hard for $k \geqslant 3$ in [2,13], and approximation algorithms for finding the anonymization that suppresses the fewest cells have been proposed in [2,13].

Recently, Machanavajjhala et al. [12] proposed the notion of ℓ-diversity as an alternative privacy requirement to k-anonymity. Li et al. [11] addressed the limitations of ℓ-diversity and proposed the notion of t-closeness as a new privacy requirement. The generalization schemes for k-anonymity discussed in this paper can be adapted for ℓ-diversity or t-closeness.

7. Conclusions

In this paper, we introduce three new generalization schemes for k-anonymity and present a taxonomy of generalization schemes. We give enumeration algorithms for the new generalization schemes and provide pruning rules and techniques to search for the optimal anonymization using discernibility metric in [4,8] and the new metric we proposed in Section 4.1. We compared the efficiency and data quality of the generalization schemes, the two approaches (bottom-up and top-down), and the effectiveness of pruning techniques through experiments on a real census data.

References

[1] C. Aggarwal, On k-anonymity and the curse of dimensionality, in: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), 2005.

[2] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, A. Zhu, Anonymizing tables, in: Proceedings of International Conference on Database Theory (ICDT), 2005.

[3] G. Aggarwal, T. Feder, K. Kenthapadi, S. Khuller, R. Panigrahy, D. Thomas, A. Zhu, Achieving anonymity via clustering, in: Proceedings of the 25th ACM Symposium on Principles of Database Systems (PODS), 2006.

[4] R. J. Bayardo, R. Agrawal, Data privacy through optimal k-anonymization, in: Proceedings of the 21st International Conference on Data Engineering (ICDE), 2005.

[5] J. Domingo-Ferrer, V. Torra, Disclosure protection methods and information loss for microdata, in: Confidentiality, Disclosure and Data Access: Theory and Practical Applications for Statistical Agencies, 2001, pp. 91–110.

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

Please cite this article in press as: T. Li, N. Li, Towards optimal k-anonymization, Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.06.015

[7] B.C.M. Fung, K. Wang, P.S. Yu, Top-down specialization for information and privacy preservation, in: Proceedings of the 21st International Conference on Data Engineering (ICDE), 2005.

[8] V. S. Iyengar, Transforming data to satisfy privacy constraints, in: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2002.

[9] K. LeFevre, D. DeWitt, R. Ramakrishnan, Incognito: efficient full-domain k-anonymity, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), 2005.

[10] K. LeFevre, D. DeWitt, R. Ramakrishnan, Mondrian multidimensional k-anonymity, in: Proceedings of the 22nd International Conference on Data Engineering (ICDE), 2006.

[11] N. Li, T. Li, S. Venkatasubramanian, t-closeness: privacy beyond k-anonymity and ℓ-diversity, in: Proceedings of the 23rd International Conference on Data Engineering (ICDE), 2007.

[12] A. Machanavajjhala, J. Gehrke, D. Kifer, M. Venkitasubramaniam, ℓ-diversity: privacy beyond k-anonymity, in: Proceedings of the 22nd International Conference on Data Engineering (ICDE), 2006.

[13] A. Meyerson, R. Williams, On the complexity of optimal k-anonymity, in: Proceedings of the 23rd ACM Symposium on Principles of Database Systems (PODS), 2004.

[14] G.C. Rota, The number of partitions of a set, American Mathematical Monthly 71 (5) (1964) 498–504.

[15] R. Rymon, Search through systematic set enumeration, in: Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR-02), 1992.

[16] P. Samarati, Protecting respondents privacy in microdata release, IEEE Transaction on Knowledge and Data Engineering 13 (6) (2001) 1010–1027.

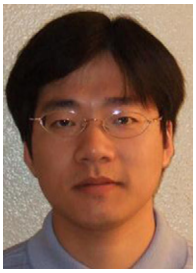
[17] P. Samarati, L. Sweeney, Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression, Technical Report SRI-CSL-98-04, SRI Computer Science Laboratory, 1998.

[18] L. Sweeney, Achieving k-anonymity privacy protection using generalization and suppression, International Journal on Uncertainty, Fuzziness and Knowledge-based Systems 10 (5) (2002) 571–588.

[19] L. Sweeney, k-anonymity: a model for protecting privacy, International Journal on Uncertainty, Fuzziness and Knowledge-based Systems 10 (5) (2002) 557–570.

[20] K. Wang, P.S. Yu, S. Chakraborty, Bottom-up generalization: a data mining solution to privacy protection, in: Proceedings of the 4th International Conference on Data Mining (ICDM), 2004.

[21] G.I. Webb, Opus: an efficient admissible algorithm for unordered search, Journal of Artificial Intelligence Research (1995) 431–465.



Tiancheng Li received his B.E. degree in Computer Science from Zhejiang University in 2005. He is currently a Ph.D. candidate in the Computer Science Department at Purdue University. His research interests include data privacy and database security.

Ninghui Li received his B.E. degree in Computer Science from the University of Science and Technology of China in 1993, and the M.Sc. and Ph.D. degrees in Computer Science from New York University, in 1998 and 2000. He is currently an Assistant Professor in Computer Science at Purdue University. Prior to joining Purdue University in 2003, he was a Research Associate at Stanford University Computer Science Department. Dr. Li's research interests are in security and privacy in information systems, with a focus on access control. He has worked on projects on trust management, automated trust negotiation, role-based access control, online privacy protection, privacy-preserving data publishing, and operating system access control. He has published more than 50 technical papers in refereed journals and conference proceedings and has served on the Program Committees of more than three dozen international conferences and workshops. He is a member of the ACM, the IEEE, the IEEE Computer Society and the USENIX Association.

Please cite this article in press as: T. Li, N. Li, Towards optimal k-anonymization, Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.06.015