Optimal k-Anonymity with Flexible Generalization Schemes through Bottom-up Searching

Tiancheng Li Ninghui Li
CERIAS and Department of Computer Science, Purdue University
250 N. University Street, West Lafayette, IN 47907-2066
{li83, ninghui}@cs.purdue.edu

Abstract

In recent years, a major thread of research on k-anonymity has focused on developing more flexible generalization schemes that produce higher-quality datasets. In this paper we introduce three new generalization schemes that improve on existing schemes, as well as algorithms enumerating valid generalizations in these schemes. We also introduce a taxonomy for generalization schemes and a new cost metric for measuring information loss. We present a bottom-up search strategy for finding optimal anonymizations. This strategy works particularly well when the value of k is small. We show the feasibility of our approach through experiments on real census data.

1. Introduction

Organizations, industries and governments are increasingly publishing microdata. 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 [6, 5, 7, 8] 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. They suggested the use of generalization and suppression for anonymizing data. 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 developing flexible generalization schemes for k-anonymity. Each generalization scheme defines a space of valid generalizations. A more flexible scheme allows a larger space of valid generalizations and can thus produce better-quality data. Many of the existing generalization schemes use value generalization hierarchies (VGHs) of attributes. In a VGH, leaf nodes correspond to actual attribute values, and internal nodes represent more-general values. Figure 1 shows a VGH for the *work-class* attribute. Three generalization schemes have been proposed and studied in the literature:

Basic Hierarchical Scheme (BHS) Earlier work on k-anonymity focuses on this scheme [3, 5, 6, 7, 8]. In BHS, all values are generalized to the same level of VGH. Thus the number of valid generalizations for an attribute is the height of the VGH for that attribute. BHS is likely to suffer from high information loss due to unnecessary generalizations, which motivated the development of other more flexible generalization schemes.

Group Hierarchical Scheme (GHS) Proposed by Iyengar [2], GHS allows different values of an 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 in that set (the value corresponding to the leaf will be generalized to the value represented by that node). GHS is likely to produce better-quality anonymizations than BHS. However, the solution space is still limited by the VGHs, and the quality of the resulted dataset depends on the choice of the VGHs.

Ordered Partitioning Scheme (OPS) The fact that the quality of the resulted dataset depends on the choice of VGHs motivated the OPS scheme [1]. OPS does not require predefined VGHs. Instead, a total order is defined over each attribute domain, and any order-preserving partition (i.e., no two blocks in the partition overlap) is a valid generalization. The total number of valid generalizations for an attribute with n values is 2^{n-1}.

The work in this paper is motivated by three observations. First, OPS requires a total order on the attribute domain. However, it is difficult to define a total order for a categorical attribute and such a total order limits the possible

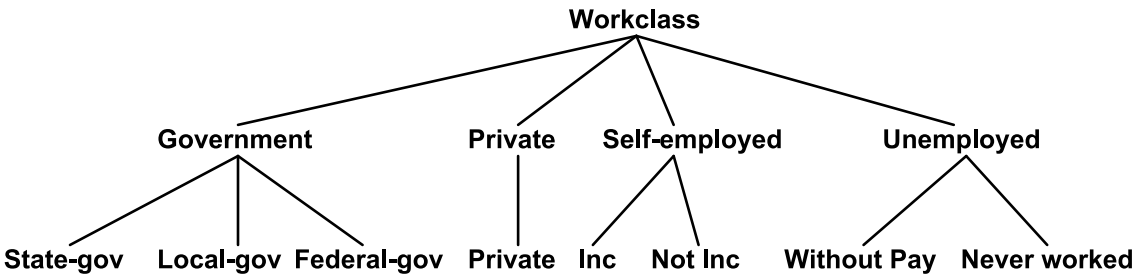


Figure 1. A value generalization hierarchy for the attribute *work-class*.

solutions. For example, consider the *work-class* attribute in Figure 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 OPS.

Second, semantic information represented by VGHs can be utilized in defining valid generalizations. Again consider Figure 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 [1] 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.

The contributions of this paper are as follows. We introduce three new generalization schemes, and compare them with existing schemes in the framework of a taxonomy tree. We also develop an approach for systematically enumerating all partitions in an unordered set. This enumeration algorithm can be used and adapted in enumerating valid generalizations for the new schemes. We also introduce a bottom-up search strategy that works better for smaller k. Finally, we introduce a fine-grained cost metric which we believe better captures information loss.

2. A Taxonomy of Generalization Schemes

Before we describe the new generalization schemes, we introduce some basic concepts. 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* $\{g_1, ..., g_m\}$ such that there is one attribute generalization for each attribute in the quasi-identifier. A tuple $t = (v_1, ..., v_m)$ in D is transformed into a new tuple $t' = (g_1(v_1), ..., g_m(v_m))$.

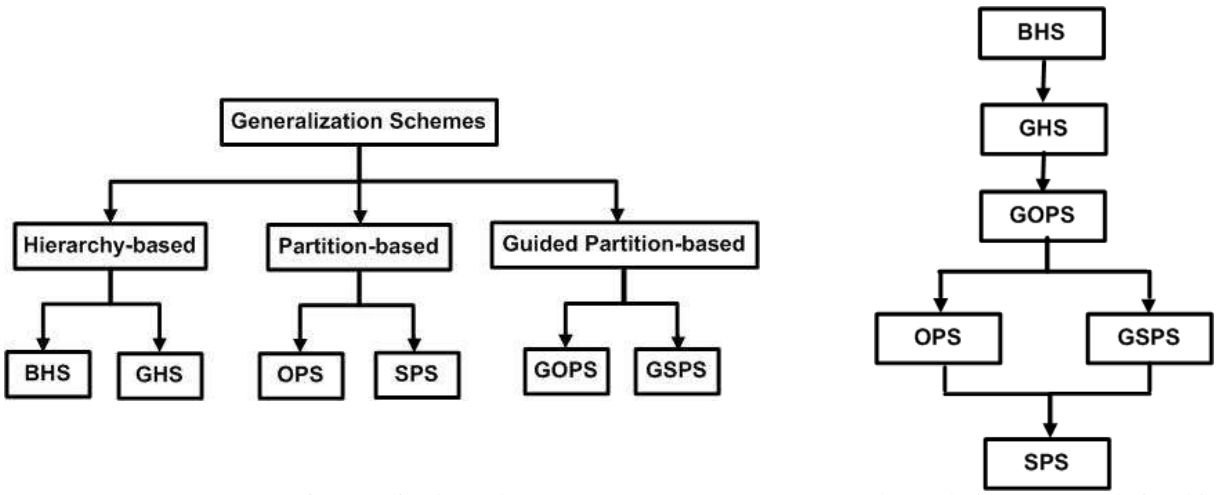
Another anonymization technique is *tuple suppression*, which removes the entire record from the table. Tuple suppression can be incorporated into the framework of generalization by first transforming the dataset D into a new dataset D' using anonymization g and then deleting any tuples in D' that fall 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. We now introduce three new generalization schemes:

Set Partitioning Scheme (SPS) OPS requires a predefined total order over the attribute domain, which is difficult to define for categorical attributes and unnecessarily imposes constraints on the space of valid generalizations. 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. We discuss how to enumerate all valid generalizations in SPS in Section 3.

The number of different ways to partition a set with n elements is known as the Bell number [4]. They satisfy the recursion formula: $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$. The first few Bell numbers are: $B_0 = B_1 = 1, B_2 = 2, B_3 = 5, B_4 = 15, B_5 = 52, ...$ There are $B_8 = 4140$ generalizations for the *work-class* attribute shown in Figure 1, as compared to 128 generalizations in OPS.

Guided Set Partitioning Scheme (GSPS) SPS does not take into account the semantic relationship among values of an attribute domain. For example, when compared with *Private*, the values *Local-gov* and *Federal-gov* are semantically closer with *State-gov*.

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



(a) A taxonomy of generalization schemes (b) "Solution space" relationship

Figure 2. A Taxonomy of generalization schemes and "solution space" relationship

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 semantical distance to x as y does not need to be in the same partition. Consider the VGH for *work-class* attribute shown in Figure 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., the VGH has only two levels: one root at the root level and all values at the leaf level.

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 the degenerated VGH is used.

Figure 2(a) 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 π_1 and π_2, we write $\pi_1 \prec \pi_2$ when the space of valid generalizations allowed by π_1 is a proper subset of the space of valid generalizations allowed by π_2. We then have the following relationship: BHS$\prec$GHS$\prec$GOPS, GOPS$\prec$OPS$\prec$ SPS, and GOPS$\prec$GSPS$\prec$SPS. The partial order relationship among the six generalization schemes is shown in Figure 2(b).

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 first study how to enumerate all generalizations 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 $S_1, S_2, ..., S_m$, such that $\Sigma = S_1 \cup S_2 \cup ... \cup S_m$. 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. Two sets S_j and S_i ($j > i$) can be merged if and only if *all three* of the following conditions are satisfied:

1. S_i contains a single element e.
2. Each set in between (i.e., $S_{j+1}, ..., S_{i-1}$) contains a single element.
3. Each element in S_j is less than e.

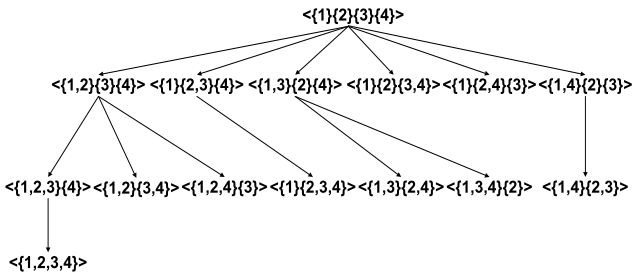


Figure 3. Enumeration tree over {1,2,3,4}

For example, the partition enumeration tree for the unordered alphabet {1,2,3,4} is shown in Figure 3.

The algorithm for enumerating all child partitions of a given partition is given in Figure 4.

Child_Partitions
Input: A partition P=$\langle S_1, ..., S_t\rangle$ on Σ
Output: The set of child partitions of P
$T = \emptyset$
for i=2 to t **do**
  if S_i contains a single element e **then**
    for j=i-1 to 1 **do**
      if e is larger than 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 contains more than one element **then break**
return T

Figure 4. Algorithm for enumerating child partitions of a given partition

The following theorem shows that our algorithm enumerates all partitions of S in a systematic manner.

Theorem: *The algorithm in Figure 4 enumerates all partitions of S in a systematic manner, i.e., each partition of S is enumerated exactly once.*

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

In order to make the proof concise, we denote "merging e with 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:
$\langle a_{12}, \{a_{11}\}\rangle$,$\langle a_{13}, \{a_{11}, a_{12}\}\rangle$,...,$\langle a_{1t_1}, \{a_{11}, a_{12}, ...a_{1t_1-1}\}\rangle$,
$\langle a_{22}, \{a_{21}\}\rangle$,$\langle a_{23}, \{a_{21}, a_{22}\}\rangle$,...,$\langle a_{2t_2}, \{a_{21}, a_{22}, ...a_{2t_2-1}\}\rangle$,
...,
$\langle a_{s2}, \{a_{s1}\}\rangle$,$\langle a_{s3}, \{a_{s1}, a_{s2}\}\rangle$,...,$\langle a_{st_s}, \{a_{s1}, a_{s2}, ...a_{st_s-1}\}\rangle$.

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 also show that the above ordering is unique through two observations:

1. a_{ij} must be merged before a_{ik} for any $i = 1, 2, ..s$ and $1 \le j < k \le t_i$. 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 $1 \le i < j \le s$, $1 < p \le t_i$ and $1 < q \le t_j$. 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}.

We have shown that our algorithm enumerates all partitions on S and each partition is enumerated exactly once. The enumeration algorithm is thus systematic.

We now study how to enumerate anonymizations for SPS. Recall that an anonymization is a set of attribute generalizations $\{P_1, P_2, ..P_m\}$ consisting of one attribute generalization per attribute. We build an enumeration tree to enumerate all valid 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.

The algorithm for enumerating all child anonymizations of a given anonymization in SPS is shown in Figure 5.

The enumeration algorithm for SPS discussed above 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

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 the applicator set of N'.
    $T = T \cup N'$
return T

Figure 5. Algorithm for enumerating child nodes of a give node

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 above. If such merging is allowed, then a new node is created. This enumeration approach remains systematic and complete.

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

4. Cost Metrics and Cost-based Pruning

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) [1]. DM 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), because the tuple needs to be suppressed.

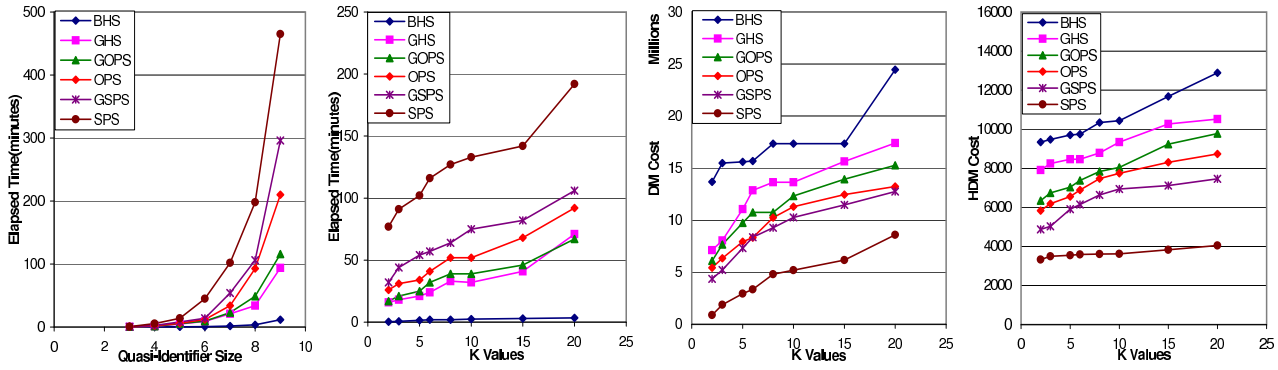
DM measures the discernibility of a record as a whole. We propose *Hierarchical Discernibility Metric (HDM)*, which captures the notion of *discernibility* among attribute values. For example, consider the *work-class* attribute in Figure 1, suppose 50 records have value *Inc* and 200 records have value *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 record is the average penalty for each attribute.

Using the cost metrics, we can compare the data quality of a dataset produced by an anonymization. Significant performance improvement can be achieved if we can effectively prune parts of the enumeration tree that will not produce an optimal solution. In [1], Bayardo and Agrawal identified a number of pruning rules using a branch and bound approach. 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. 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. Other pruning rules include useless value pruning, where applicators that cannot produce better anonymizations are removed from the applicator set.

In addition to the pruning rules employed by Bayardo and Agrawal [1], we also use inclusive pruning rules introduced by Webb [9]; these rules identify applicators that must be included in order to produce the best anonymization. This class of pruning rules can be effective if we require all records satisfy k-anonymity requirement and no suppression is allowed.

Besides these pruning rules, we used a number of techniques to facilitate finding the optimal solution. One technique we used is a modified BFS search strategy. Due to the imprecise estimation of the best cost, we could do a lot of unnecessary search if simple BFS is used. 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 has 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.

Using these techniques, we showed the feasibility of our generalization schemes through experiments on real census dataset.



(a) Performance ($k = 5$) (b)Performance ($QI = 7$) (c) Data quality (DM) (d) Data Quality (HDM)

Figure 6. Comparison of the six generalization schemes in terms of performance and data quality

5. Experiments

The dataset used in the experiments is the adult dataset from the UC Irvine machine learning repository, as described in Figure 7. The algorithms are implemented in JAVA and experiments are run on a 3.4GHZ Pentium 4 machine with 2GB Physical Memory Space.

	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

Figure 7. Description of the *Adult* dataset

We first compare the performance of the generalization schemes, as shown in Figure 6(a) and (b). The running time increases as we use a larger quasi-identifier. 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 a generalization scheme increases as k increases. The data reported in [1] 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.

We compare the data quality of the resulted dataset produced by the six generalization schemes, as shown in Figure 6(c) and (d). 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.

Acknowledgement

Portions of this work were supported by NSF IIS-0430274 and sponsors of CERIAS. We thank the anonymous reviewers for their helpful comments.

References

[1] R. J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, pages 217–228, 2005.

[2] V. S. Iyengar. Transforming data to satisfy privacy constraints. In *SIGKDD*, pages 279–288, 2002.

[3] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *SIGMOD*, pages 49–60, 2005.

[4] G.-C. Rota. The number of partitions of a set. In *Amer. Math. Monthly*, 71(5):498–504, 1964.

[5] P. Samarati. Protecting respondent's privacy in microdata release. In *IEEE TKDE*, 13(6):1010–1027, 2001.

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

[7] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. In *Int. J. on Uncertain. Fuzz*, 10(6):571–588, 2002.

[8] L. Sweeney. K-anonymity: A model for protecting privacy. In *Int. J. on Uncertain. Fuzz*, 10(5):557–570, 2002.

[9] G. I. Webb. Inclusive pruning: a new class of pruning rule for unordered search and its application to classification learning. In *Proc. Australasian Computer Science Conference*, pages 1–10, 1996.