Data Privacy Through Optimal k-Anonymization

Roberto J. Bayardo
IBM Almaden Research Center
bayardo@alum.mit.edu

Rakesh Agrawal
IBM Almaden Research Center
rakesh_agrawal@ieee.org

Abstract

Data de-identification reconciles the demand for release of data for research purposes and the demand for privacy from individuals. This paper proposes and evaluates an optimization algorithm for the powerful de-identification procedure known as k-anonymization. A k-anonymized dataset has the property that each record is indistinguishable from at least $k-1$ others. Even simple restrictions of optimized k-anonymity are NP-hard, leading to significant computational challenges. We present a new approach to exploring the space of possible anonymizations that tames the combinatorics of the problem, and develop data-management strategies to reduce reliance on expensive operations such as sorting. Through experiments on real census data, we show the resulting algorithm can find optimal k-anonymizations under two representative cost measures and a wide range of k. We also show that the algorithm can produce good anonymizations in circumstances where the input data or input parameters preclude finding an optimal solution in reasonable time. Finally, we use the algorithm to explore the effects of different coding approaches and problem variations on anonymization quality and performance. To our knowledge, this is the first result demonstrating optimal k-anonymization of a non-trivial dataset under a general model of the problem.

1. Introduction

Industries, organizations, and governments must satisfy demands for electronic release of information in addition to demands of privacy from individuals whose personal data may be disclosed by the process. As argued by Samarati and Sweeney [11], naive approaches to de-identifying microdata are prone to attacks that combine the data with other publicly available information to re-identify represented individuals. For example, consider a dataset of patient diagnoses that has been "scrubbed" of any personal identifiers such as name or social security number. While no record contains any single identifying value, many records are likely to contain unique value combinations. Imagine for instance a represented individual who is the only male born in 1920 living in some sparsely populated area. This individual's age, gender, and zip code could be joined with a voter registry from the area to obtain his name, revealing his medical history.

To avoid such so-called linking attacks while preserving the integrity of the released data, Samarati and Sweeney have proposed the concept of k-anonymity [11]. A k-anonymized dataset has the property that each record is indistinguishable from at least $k-1$ other records within the dataset. The larger the value of k, the greater the implied privacy since no individual can be identified with probability exceeding $1/k$ through linking attacks alone.

The process of k-anonymizing a dataset involves applying operations to the input dataset including data suppression and cell value generalization. Suppression is the process of deleting cell values or entire tuples. Generalization involves replacing specific values such as a phone number with a more general one, such as the area code alone. Unlike the outcome of other disclosure protection techniques that involve condensation [1], data scrambling and swapping [6,7], or adding noise [2], all records within a k-anonymized dataset remain truthful.

De-identifying data through common formulations of k-anonymity is unfortunately NP-hard if one wishes to guarantee an optimal anonymization [8]. Algorithms that are suitable for use in practice typically employ greedy methods [6,13] or incomplete stochastic search [5,16], and do not provide any guarantees on the quality of the resulting anonymization.

We propose a practical method for determining an optimal k-anonymization of a given dataset. An optimal anonymization is one which perturbs the input dataset as little as is necessary to achieve k-anonymity, where "as little as is necessary" is typically quantified by a given cost metric. Several different cost metrics have been proposed [5,6,10,14], though most aim in one way or another to minimize the amount of information loss resulting from the generalization and suppression operations that are applied to produce the transformed dataset. The ability to compute optimal anonymizations lets us more definitively investigate the impacts of various coding techniques and problem variations on anonymization quality. It also allows us to better quantify the effectiveness of stochastic or other non-optimal methods.

We perform experiments to illustrate the feasibility of our approach. We demonstrate that despite the problem's inherent hardness, provably optimal k-anonymizations can be obtained for real census data under two representative cost metrics -- in most cases within only a few seconds or

Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)
1084-4627/05 $20.00 © 2005 **IEEE**

minutes. Some parameter settings (specifically, very small values of k) remain challenging; but even under these conditions, the algorithm can still be used to produce good solutions very quickly, and constantly improving solutions throughout its execution. This *anytime* quality allows it to be used to obtain good anonymizations even when an optimal anonymization is out of reach.

Our algorithm departs from previous proposals in a variety of ways. First, previous proposals [6,10,13] suggest starting from the original dataset and systematically or greedily generalizing it into one that is k-anonymous. Our algorithm instead starts with a fully generalized dataset (one in which every tuple is identical to every other) and systematically specializes the dataset into one that is minimally k-anonymous. While this choice may seem arbitrary, it is in fact an essential ingredient to the approach. Second, our algorithm uses a tree-search strategy exploiting both cost-based pruning and dynamic search rearrangement. These techniques have proven successful in data-mining and machine learning domains [3,9,15], but to our knowledge have not been applied to the problem of k-anonymization. Third, we propose novel data-management strategies to reduce the cost of evaluating a given anonymization. Computing the cost of a k-anonymization can involve scanning if not sorting the entire input dataset. We show how sorting and scanning costs can be dramatically reduced. Combined, this suite of techniques allow significant reduction in data management overhead as well as a taming of the combinatorial state explosion. For example, the census dataset used in our experiments has a state-space size of 2^{162}. Despite this, our algorithm identifies a provably optimal k-anonymization after exploring only on the order of up to a few hundred thousand states. Other optimal algorithms proposed in the literature are suitable only for input datasets with trivially small domains.

2. Related Work

While there are several k-anonymization algorithm proposals in the literature [5,6,8,10,12,13,16], only a few are suitable for use in practice. Iyengar [5] shows how to attack a very flexible (and highly combinatorial) formulation of k-anonymity using a genetic algorithm. The algorithm may run for hours, and because it is an incomplete stochastic search method, it cannot provide any guarantees on solution quality. The μ-argus algorithm of Hundpool and Willenborg [6] computes the frequency of all 2 and 3-value combinations of dataset values, then greedily generalizes values and suppresses outliers in order to achieve k-anonymity. The datafly approach of Sweeney [12] is another greedy approach that generates frequency lists and iteratively generalizes those combinations with less than k occurrences. Like incomplete stochastic approaches, iterative greedy approaches such as μ-argus and Datafly offer no solution quality guarantees. Sweeney [12] and Samarati [10] have both proposed complete algorithms for k-anonymization. Sweeney's algorithm exhaustively examines all potential generalizations to identify the optimal (or "preferred") generalization that minimally satisfies the anonymity requirement, acknowledging the approach is impractical even on modest sized datasets. Samarati proposes an algorithm to identify all "k-minimal" generalizations, among which reside the optimal k-anonymizations according to certain preference criteria. While the algorithm exploits binary search and a monotonicity property on the generalization lattice to avoid exhaustively searching the entire generalization space, the number of k-minimal generalizations itself remains exponential and can easily become too large to enumerate efficiently.

Winkler [16] has proposed using simulated annealing to attack the problem but provides no evidence of its efficacy. On the more theoretical side, Meyerson and Williams have recently proposed an approximation algorithm that achieves an anonymization within $O(k\log k)$ of optimal. The method remains to be tested in practice. The table below summarizes this set of known approaches to the problem in terms of practicality and solution guarantees.

Algorithm	Practical?	Guarantee
Sweeney-Datafly	**Y**	none
Sweeney-MinGen	N	**optimal**
Samarati-AllMin	N	**optimal**
Iyengar-GA	**Y**	none
Winkler-Anneal	possible	none
Meyerson-Approx	possible, but only for small k using suppression alone	$O(k\log k)$ of optimal
our proposal	**Y**	**optimal**

Table 1. A breakdown of known approaches to k-Anonymity.

3. Preliminaries

A *dataset* D is a multi-set of *tuples* $t_1 \ldots t_n$ each comprising a sequence of m values $\langle v_1, v_2, \ldots, v_m \rangle$. The set of values that may appear in position i of a tuple is called the *domain* for *attribute* i, and is denoted Σ_i.

The problem of (k-)anonymizing a dataset has been formalized in a variety of ways. In some formulations [6,8,14], anonymization is achieved (at least in part) by suppressing (deleting) individual values from tuples. In others [5,10,14], every occurrence of certain attribute values within the dataset is replaced with a more general value (for example, the zip codes 95120-95129 might be replaced with 9512*). Some generalization based formulations also allow entire tuples to be suppressed to avoid excessive generalization due to outliers [5,10].

The exact way values can be generalized also differs among formulations. Iyengar [5] provides a flexible generalization model based on imposing an order on the attribute domains (if one is not already implicit or explicit) and

allowing generalizations defined by partitionings that respect the ordering. Iyengar shows that generalization models based on value hierarchies [10,14] map to this approach, though hierarchies effectively impose additional constraints on the particular partitionings that may be considered. Flexible value generalization schemes such as Iyengar's provide more choice in anonymizing the data, which can result in better information preservation. But such choice comes at the cost of a significant increase in the number of allowable states that must be searched to identify an optimal or near-optimal solution. Nevertheless, because the value generalization formalization of Iyengar encompasses and extends many others (with the exception of those involving cell level suppression), we adopt his framework in this paper.

An *attribute generalization* for an attribute with ordered value domain Σ is a partitioning of the attribute domain into *intervals* $\langle I_1, I_2, \ldots, I_o \rangle$ such that every value of Σ appears in some interval, and every value in an interval I_j precedes every value in the intervals following I_j. For example, a generalization of an "age" attribute that ranges over ages [1],[2],..,[30] might be $\langle [1, 10], [11, 20], [21, 30] \rangle$. For purely categorical domains, the domain ordering must be supplied by the user. As Iyengar notes, should a generalization hierarchy be available, this ordering should correspond to the order in which the leaves are output by a preorder traversal of the hierarchy. An attribute generalization a can be *applied* to a value v within the domain of the attribute, denoted by $a(v)$, to return the interval in which v belongs.

We will denote each interval within an attribute generalization with the least value belonging to the interval. This allows specifying attribute generalizations much more succinctly. For example, the age generalization from the paragraph above is specified as $\{1, 11, 21\}$. Together with the ordered attribute domain, this representation is complete and unambiguous.

An *anonymization* of a dataset set D is a set of attribute generalizations $\{a_1 \ldots a_m\}$ consisting of exactly one generalization per attribute of D. These generalizations are said to *transform* D into a new dataset $D' = \langle a_1(v_{1,1}), \ldots, a_m(v_{1,m}) \rangle, \ldots, \langle a_1(v_{n,1}), \ldots, a_m(v_{n,m}) \rangle$ where value $v_{i,j}$ denotes the jth value of tuple i from dataset D. More intuitively, an anonymization injectively maps each and every value in the original dataset to a new (generalized) value. In practice a database may also consist of columns that need not be subject to generalization. For instance data that has never before been disclosed publicly cannot be used in linking with outside data sources and may in some cases be exempted from the anonymization process. We assume without loss of generality that each and every column contains potentially publicly linkable information and must therefore be generalized to achieve anonymity.

In transforming a dataset D, an anonymization is said to *induce* new equivalence classes of tuples that are defined by applying the tuple equality operator over transformed tuples. Put another way, each induced equivalence class consists of a maximal set of tuples that are equivalent within the transformed dataset.

DEF 3.1: (K-ANONYMIZATION WITHOUT SUPPRESSION) A *k-anonymization* of a dataset D is an anonymization of D such that the equivalence classes induced by the anonymization are all of size k or greater.

We also say that a (transformed) dataset is *k-anonymized* if every equivalence class of tuples is of size k or greater.

3.1 Modeling Tuple Suppression

In some cases, outliers in a dataset may necessitate significant generalization in order to achieve a k-anonymization, resulting in unacceptable information loss. To fix this situation, we may choose to s*uppress* outlier tuples by deleting them. More formally, given an anonymization g of dataset D, we say g *k-transforms* D into the dataset D'' where D'' is the result of:

(1) transforming D into D' using g as defined before, followed by,

(2) deleting any tuple from D' belonging to an induced equivalence class of size less than k.

Note then that any k-transformed dataset is always k-anonymized. An anonymization is said to *k-suppress* (or just suppress when k is clear from the context) those tuples removed from D' to form D''.

3.2 Problem Statement

The problem of k-anonymity is not simply to find any k-anonymization, but to instead find one that is "good" according to some quantifiable cost. The problem of optimal k-anonymity is to find one that is known to be "best."

More precisely, when suppression is allowed, we are searching for an anonymization that produces the "best" k-transformed dataset, as determined by some cost metric. Such an anonymization is said to be *optimal*. Since k-anonymization without suppression can be modeled in the suppression-based framework by imposing an infinite cost whenever suppression is required, from here on we focus on the case where suppression is allowed. We will note specific impacts of the suppression-free case where appropriate.

3.3 Modeling Desirable Anonymizations

Each of the works cited earlier provides its own unique metrics for modeling desirable anonymizations. Our algorithm is for the most part metric agnostic. We propose it in a manner whereby it can be instantiated to use a particular metric by defining a simple cost-bounding function. To keep the presentation concrete, we will show how cost-bounding is accomplished with two example metrics. The strategies used should be easily adaptable to other metrics, which are almost all variants of the two considered here.

Cost metrics typically tally the information loss resulting from the suppression or generalizations applied. Each anonymization can thus be thought of as imparting a "penalty" on each tuple that reflects the information loss associated with its transformation or suppression. Naturally, most

AGE GENDER MARITAL STATUS

<[10-29][30-39][40-49]> <[M][F]> <[Married][Widowed][Divorced][Never Married]>

.....**1***.....**2**.....**3**.....**4***.**5**.....**6***.....**7**.....**8**.....**9**.....

Figure 1. Example total ordering of the value domains for a simple 3-attribute/9-value table. Given a total ordering, values can be identified by their position along the ordering. The least value from each attribute domain (flagged with a *) must appear in any valid generalization and hence can be treated as implicit.

cost metrics will penalize a suppressed tuple at least as much as any generalization of the tuple.

The first metric we use is one that attempts to capture in a straightforward way the desire to maintain discernibility between tuples as much as is allowed by a given setting of k. This *discernibility metric* assigns a penalty to each tuple based on how many tuples in the transformed dataset are indistinguishable from it. If an unsuppressed tuple falls into an induced equivalence class of size j, then that tuple is assigned a penalty of j. If a tuple is suppressed, then it is assigned a penalty of $|D|$, the size of the input dataset. This penalty reflects the fact that a suppressed tuple cannot be distinguished from any other tuple in the dataset. The metric can be mathematically stated as follows:

$$C_{\mathrm{DM}}(g, k) = \sum_{\forall E \text{s.t.} |E| \geq k} |E|^2 + \sum_{\forall E \text{s.t.} |E| < k} |D||E|$$

In this expression, the sets E refer to the equivalence classes of tuples in D induced by the anonymization g. The first sum computes penalties for each non-suppressed tuple, the second for suppressed tuples.

Another interesting cost metric we use was originally proposed by Iyengar [5]. This metric can be applied when tuples are assigned a categorical class label in an effort to produce anonymizations whose induced equivalence classes consist of tuples that are uniform with respect to the class label. This *classification metric* assigns no penalty to an unsuppressed tuple if it belongs to the majority class within its induced equivalence class. All other tuples are penalized a value of 1. More precisely:

$$C_{\mathrm{CM}}(g, k) = \sum_{\forall E \text{s.t.} |E| \geq k} (|\text{minority}(E)|) + \sum_{\forall E \text{s.t.} |E| < k} |E|$$

The minority function within the above statement accepts a set of class-labeled tuples and returns the subset of tuples belonging to any minority class with respect to that set. As before, the first sum from this expression penalizes non-suppressed tuples, and the second one suppressed tuples. Iyengar has shown that this *class-conscious* metric produces anonymized datasets that yield better classification models than do class-oblivious metrics [5].

Since suppressing a tuple is a rather drastic operation, it may be desirable to impose a hard limit on the number of suppressions allowed [10]. This can be modeled within both metric expressions by adding a condition that imposes an infinite cost should the number of suppressed tuples (the sum of all equivalent class sizes that are less than k) exceed this limit.

4. A Set Representation for Anonymizations

In this section, we set up the problem by precisely defining the space of anonymizations for a given dataset as the powerset of a special alphabet. Searching for an optimal solution then reduces to identifying the subset of the alphabet that represents the anonymization with the lowest cost.

As we have noted in Section 3, a generalization of an attribute value domain can be represented as the set of values containing only the least value from each interval. Recall that an anonymization is a set of generalizations, one for each attribute column of the dataset. We can impose a total order over the set of all attribute domains such that the values in the ith attribute domain (Σ_i) all precede the values in any subsequent attribute domain (Σ_j for $j > i$). Values inside a value domain must already be ordered, and this order is preserved by the total order. The idea is best illustrated through example. Figure 1 shows the resulting total value ordering of a 3-attribute table containing a pre-partitioned Age attribute, a categorical Gender attribute, and Marital Status attribute. Given such an ordering, we can unambiguously represent an anonymization as the union of the individual generalizer sets for each attribute. Since the least value from each value domain must appear in the generalizer for that domain, these values can be omitted without causing ambiguity. In the example from Figure 1, these values are 1 (age=10-29), 4 (gender=M) and 6 (Marital Status=Married).

Let us denote the least value in attribute domain i as v_{l_i}. We now have that the powerset of the following alphabet Σ_{all} precisely defines the entire set of anonymizations of the given dataset:

$$\Sigma_{\mathrm{all}} = (\Sigma_1 \backslash v_{l_1}) \cup (\Sigma_2 \backslash v_{l_2}) \cup \ldots \cup (\Sigma_m \backslash v_{l_m})$$

Conveniently, the empty set { } always represents the *most general* anonymization in which the induced equivalence classes consist of only a single equivalence class of identical tuples. The set Σ_{all} represents the *most specific* anonymization -- it transforms the dataset into one identical to the original dataset. Given an anonymization, adding a new value further *specializes* the data, and removing some value *generalizes* it.

Example: Consider the anonymization {2, 7, 9} defined by the value ordering in Figure 1. After adding in the

implicit least values, we have that this anonymizer represents the following generalizations: $\{1, 2\}$, $\{4\}$, and $\{6, 7, 9\}$. These in turn represent the following value intervals used for transforming the dataset: Age: <[10-29], [30-49]>, Gender: <[M or F]>, Marital status: <[Married], [Widowed or Divorced], [Never Married]>

5. A Systematic Search Strategy

We have framed the problem of identifying an optimal k-anonymization as one involving searching through the powerset of Σ_{all} for the anonymization with lowest cost. Searching the set of all subsets of a given alphabet is a problem to which various generic search methods may be applied. Our decision to frame the problem as a powerset search problem was motivated by the success of these methods in solving other NP-hard problems such as (maximal) itemset mining and rule induction [3,9,15].

One such generic search method is the OPUS framework [15]. OPUS extends a systematic set-enumeration-search strategy [9] with dynamic tree rearrangement and cost-based pruning for solving optimization problems. The set-enumeration search strategy is a straightforward method of systematically enumerating all subsets of a given alphabet through tree expansion. For example, the set-enumeration tree for the alphabet $\{1, 2, 3, 4\}$ is displayed in Figure 2.

Any node in a set enumeration tree can be represented by the set enumerated by the node, which we call the *head* set. In the context of k-anonymization, the head set represents an anonymization whose cost is to be tested. We call the set of values that can be appended to the head set to form a child the *tail* set. The tail set is actually an ordered set; the ordering specifies in what order children are to be expanded. For the example tree in the figure, the tail ordering is lexicographic. Note that the first child of a node inherits all but the first (unpruned) tail values from its parent. Subsequent children inherit one fewer tail value until the last (right-most) child, which inherits no tail values, and hence has no children of its own. We call the set formed by taking the union of the head and tail sets the *allset* of that node. The allset represents the "most specific" state that can be enumerated by a node's descendants. Figure 3 annotates each node from the tree in Figure 2 with the ordered tail sets. It also illustrates the effect of pruning a tail value, which we will elaborate on in the following subsection.

Naively, we can search for the optimal anonymization by fully traversing the set-enumeration tree over Σ_{all} using some standard traversal strategy such as depth-first search. At each node in the tree, the cost of the anonymization represented by the head set is computed and compared against the best cost anonymization identified so far. If the cost is lower than the best cost found so far, the anonymization and its cost are retained. Since set enumeration is systematic and complete, once the algorithm terminates, we are guaranteed to have identified an optimal anonymization.

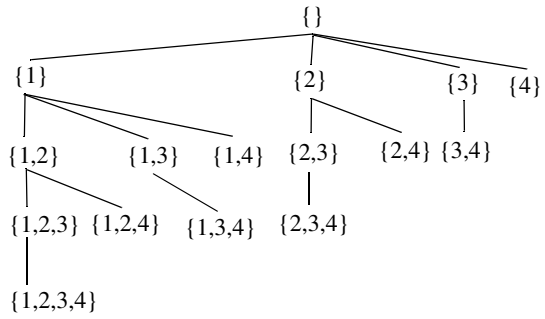

Figure 2. Set enumeration tree over alphabet {1,2,3,4}.

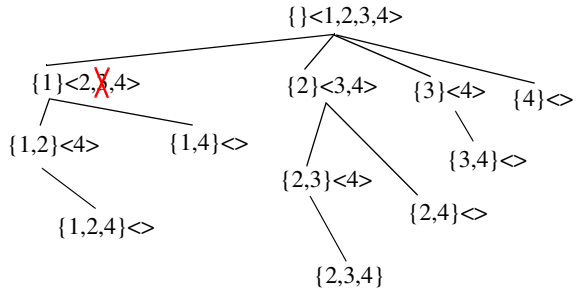

Figure 3. Set enumeration tree with explicit tail set representation, depicting the effect of pruning value 3 from the tail of node $\{1\}$

Unfortunately, the powerset of an alphabet Σ is of size $2^{|\Sigma|}$. This large a state space implies that for such an algorithm to be practical, heuristics and admissible pruning strategies must be applied to keep the expected runtime from approaching the worst case. The following sections describe the specific pruning and heuristic reordering strategies we have devised.

5.1 Pruning Overview

At each node in the search tree, our algorithm first tries to prune the node itself. Failing that, the algorithm attempts to prune values from the tail set of the node. Pruning tail values when the node itself cannot be pruned may markedly reduce the search space since it reduces the branching factor of the node and all its children that would have otherwise inherited the pruned value. For example, consider the result of pruning the value 3 from the tail of node $\{1\}$ from the example tree (Figure 3). In this example, removing value 3 prunes four nodes. More subtly, pruning a tail value can improve cost lower-bounding, leading to additional pruning beneath the node.

To guarantee optimality, a node can be pruned only when the algorithm can precisely determine that none of its descendants could be optimal. This determination can be made by computing a lower-bound on the cost achievable by any node within the subtree rooted beneath it. If the lower-bound exceeds the current best cost, the node is

pruned. For pruning tail values, we apply the same bounding function but to an "imaginary" node as follows: given a node H with tail set T, if the imaginary node consisting of head set $H \cup \{v\}$ and tail set $T \backslash v$ for some $v \in T$ is prunable according to our bounding function, then v can be removed from the tail set T of node H without affecting search completeness [3]. Pruning tail values reduces the size of a node's allset. Because allset size often impacts the quality of the cost lower-bound, it can be advantageous to recompute lower-bounds should the allset change. Our implementation recursively attempts to prune tail values whenever the allset is reduced.

5.2 Computing Cost Lower-Bounds

Cost lower-bounds must be computed in a way that is specific to the particular cost metric being used. However, the methods we use to obtain bounding functions are general, and can be easily applied to most metrics in the literature.

An excellent starting point to lower-bounding costs involves exploiting the fact that tuple suppression costs increase monotonically as we descend into the tree. By virtue of the tree expansion policy which appends values to the head set H when expanding a child of H in the search tree, the set H represents the most general anonymization that will be considered by the current node H and any of its descendants. This implies that if the anonymization H suppresses a set of tuples S, then the set of tuples suppressed by any descendant is guaranteed to contain S. Stated more formally:

OBSERVATION 5.1: At a node H in the search tree, the set of tuples suppressed by the anonymization H is a subset of the set of tuples suppressed by any descendant of H.

Given this fact, we can compute a lower-bound on the cost achievable by any node in the subtree rooted at H by totaling the suppression cost that will be imparted by the tuples S suppressed by H. For the discernibility metric, this is given by the value $|D| \cdot |S|$. For the classification metric, the total penalty is $|S|$. For the variants involving hard limits on the number of suppressed tuples, when $|S|$ exceeds the limit, the penalty is instead ∞.

Lower-bounding by totaling the cost of suppressed tuples is a good start, but it's possible to improve the lower-bound by also adding in a lower-bound on the penalty resulting from tuples that aren't suppressed by H. One way to do this is to consider the effect of the allset anonymization. Recall that the allset is the union of the head and tail sets of a given node, and therefore represents an anonymization that is strictly more specific than any other anonymization to be considered within the current subtree. Given this, we note the following fact:

OBSERVATION 5.2: For a node H with allset A, the equivalence classes induced by A are subsets of the equivalence classes induced by any descendant of H.

This observation lets us lower-bound the size of an equivalence class which a tuple will fall into. Let us now consider the impacts of this observation on each cost metric individually. For the discernibility metric, the observation implies that a tuple t that is not suppressed by H must be penalized at least the value $|E|$ where E is the equivalence class induced by A and contains t. This lower-bound is as good as we can do if $|E| \geq k$. If t falls into an induced equivalence class of A that is smaller than k, then the penalty imparted on it by some descendant of H will be either $|D|$ (for nodes where t is suppressed) or some value that is at least k (since unsuppressed tuples are always penalized a value of k or higher). We have now determined a lower-bound on the cost imparted by every tuple under the discernibility metric. For a tuple t, let us denote the equivalence class induced by the allset A that contains t as $E_{A,t}$. We can state the discernibility metric lower-bound formally as follows:

$$LB_{\mathrm{DM}}(H, A) = \sum_{\forall t \in D} \begin{cases} |D| & \text{when } t \text{ is suppressed by } H, \\ \max(|E_{A,t}|, k) & \text{otherwise} \end{cases}$$

For the classification metric, instead of penalizing on a tuple by tuple basis, we will bound the cost by penalizing each of the allset induced equivalence classes. Recall that for a set of class labeled tuples, the minority( ) function returns the tuples belonging only to the minority classes with respect to that set. The majority() function can be defined analogously as returning only those tuples belonging to the majority class. Given this notation, we prove the following claim:

CLAIM 5.3: For any two sets of class labeled tuples E_1 and E_2, the following statement holds: $|\mathrm{minority}(E_1 \cup E_2)| \geq |\mathrm{minority}(E_1)| + |\mathrm{minority}(E_2)|$.

Proof: Note that

$$E_1 \cup E_2 = \mathrm{minority}(E_1) \cup \mathrm{minority}(E_2) \cup \mathrm{majority}(E_1) \cup \mathrm{majority}(E_2)$$

To illustrate the proof technique, let us first consider the case where there are only 2 classes. In such a case, the contribution of E_1 to the minority class of $E_1 \cup E_2$ is either $\mathrm{minority}(E_1)$ or $\mathrm{majority}(E_1)$, and the contribution of E_2 to the minority class of $E_1 \cup E_2$ is either $\mathrm{minority}(E_2)$ or $\mathrm{majority}(E_2)$. The claim follows then from the fact that when there are only two classes, $|\mathrm{majority}(E)| \geq |\mathrm{minority}(E)|$ for any set of tuples E.

For the case where there are more than 2 classes, we have that the contribution of E_1 to the minority class of $E_1 \cup E_2$ is either $\mathrm{minority}(E_1)$ or it is $\mathrm{majority}(E_1) \cup \{\mathrm{minority}(E_1) - E_{1,c}\}$ for some set $E_{1,c}$. More precisely, we know that $E_{1,c}$ is the set of tuples in E_1 that have the same class label as the majority class in $E_1 \cup E_2$. Clearly, $E_{1,c}$ is a subset of $\mathrm{minority}(E_1)$ and therefore disjoint from $\mathrm{majority}(E_1)$. Furthermore, we know that $|E_{1,c}| \leq |\mathrm{majority}(E_1)|$. We thus have that:

Figure 4. A sorted dataset with its tuple equivalence classes demarcated. Dashed lines illustrate the splitting effect on equivalence classes resulting from adding a single value to the represented anonymization. If the new equivalence classes created by these splits are all of size less than k ,then the value is called a *useless* value.

$$\begin{aligned}&\left|\text{majority}(E_1) \cup \text{minority}(E_1) - E_{1,c}\right| \geq \\ &\left|\text{majority}(E_1)\right| + \left|\text{minority}(E_1)\right| - \left|E_1\right| \geq \\ &\left|\text{majority}(E_1)\right| + \left|\text{minority}(E_1)\right| - \left|\text{majority}(E_1)\right| = \\ &\left|\text{minority}(E_1)\right|\end{aligned}$$

We repeat the exact argument with E_2 in place of E_1 to show that its contribution to the minority class of $E_1 \cup E_2$ is at least $\left|\text{minority}(E_2)\right|$, and the claim follows. □

This fact leads to the following classification metric lower-bound for a node H with allset A :

$$LB_{\text{CM}}(H, A) = \sum_{\forall E \text{ induced by } A} \begin{cases} |E| & \text{when } E \text{ is suppressed by } H, \\ |\text{minority}(E)| & \text{otherwise} \end{cases}$$

In the equation above, when an induced equivalence class E is suppressed by H, we mean that the tuples in E are suppressed by anonymization H. By Observation 5.2, either all tuples in an equivalence class E induced by the allset are suppressed by H, or none of them are.

5.3 Useless Value Pruning

Another pruning technique we use is to prune tail values representing specializations that can be easily determined to have no hope of improving anonymization cost. Before formally defining this concept, we illustrate the effect of specialization on the induced equivalence classes of an anonymization. Recall that descending to a child of any node involves adding exactly one tail value to the anonymization. This new value will "split" some interval of the anonymization into two new intervals. The impact on the node's induced equivalence classes is that some of them will split into two new classes, with the rest remaining unaffected, as illustrated in Figure 4.

Example: Consider again the example dataset consisting of age, gender, and marital status. Suppose we go from the root node ($\{\ \}$) to the child node $\{5\}$. The root node induces a single equivalence class consisting of all tuples (since all columns are fully generalized). The child $\{5\}$ corresponds to specializing on the gender attribute since the value "F" for Female is the fifth value along the total order. The single equivalence class of the parent node is thus split into two equivalence classes, one consisting of all males, and one consisting of all females.

Note that specializing doesn't typically split all existing equivalence classes. For instance, we may split an existing age interval consisting of ages from $[1-50]$ into the ranges $[1-25]$ $[26-50]$. In this case, all equivalence classes consisting of tuples with ages in the range of 50 and up will not be affected by the specialization.

More formally now, given a node H, T and some value v from T, specializing H with value v results in splitting some (typically proper) subset of H-induced equivalence classes into new equivalence classes. If these new equivalence classes are all less than size k, then the effect of the specialization is only to suppress tuples without changing the status of those that remain unsuppressed. Furthermore this effect of specializing on v holds for any descendant of H, T in the tree. We call any such tail value v of a node H, T whose only effect is to split induced equivalence classes into only suppressed equivalence classes a *useless* value with respect to H, T (see Figure 4 and its caption for an illustration.)

CLAIM 5.4: Consider the set of descendants of a given node H, T whose anonymizations have the best cost among all other descendants of H, T. If the cost metric always penalizes suppression of a tuple at least as high as any generalization, then there exists a member of this set whose anonymization contains no useless values with respect to H, T.

Proof: We prove the claim by showing that if there exists a best-cost descendant containing useless values, then removing the useless values must produce an anonymization of equivalent cost. Since this new anonymization is itself a descendant of H, T, the claim follows.

If a best cost descendant anonymization H' contains a useless value v, consider the anonymization $H'\backslash v$ formed by removing v from H'. By definition of useless value, this anonymization induces a set of equivalence classes identical to those of H' except that some equivalence classes suppressed by H' are merged. Recall that we assume only that the metric assigns a penalty to any suppressed tuple that is at least as high as the penalty assigned to any generalization of the tuple. Of those tuples affected by removing v from the anonymization, some may remain suppressed (in which case the penalty remains unchanged) and others may become unsuppressed (in which case the penalty is unchanged or may decrease). The anonymization $H'\backslash v$ must therefore

```
K-OPTIMIZE(k, head set H, tail set T, best cost c)
    ;; This function returns the lowest cost of any
    ;; anonymization within the sub-tree rooted at
    ;; H that has a cost less than c (if one exists).
    ;; Otherwise, it returns c.
    T ← PRUNE-USELESS-VALUES(H, T)
    c ← MIN(c, COMPUTE-COST(H))
    T ← PRUNE(H, T, c)
    T ← REORDER-TAIL(H, T)
    while T is non-empty do
        v ← the first value in the ordered set T
        H_new ← H ∪ {v}
        T ← T − {v}        ;; preserve ordering
        c ← K-OPTIMIZE(k, H_new, T, c)
        T ← PRUNE(H, T, c)
    return c
```

```
PRUNE(k, head set H, tail set T, best cost c)
    ;; This function creates and returns a new
    ;; tail set by removing values from T that
    ;; cannot lead to anonymizations with cost
    ;; lower than c
    if COMPUTE-LOWER-BOUND(k, H, H ∪ T) ≥ c
        then return ∅
    T_new ← T
    for each v in T do
        H_new ← H ∪ {v}
        if PRUNE(H_new, T_new − {v}, c) = ∅
            then T_new ← T_new − {v}
    if T_new ≠ T then return PRUNE(H, T_new, c)
    else return T_new
```

Figure 5. The K-OPTIMIZE procedure for computing the cost of an optimal k-anonymization.

have a cost that is at least as good as that of H'. Since H' is known to be of best cost among all of the descendants of H, T, the cost of $H'\backslash v$ must in fact be equivalent to that of H'. This process of removing useless values can be repeated to create a best-cost anonymization without any useless values, thereby proving the claim. □

A simple corollary of this claim is that any useless value can be immediately pruned from the tail set without compromising the algorithm's ability to discover an optimal solution.

We note that the definition of useless value can in many cases be broadened if we are willing to assume additional properties of the cost metric. Consider a value v such that for any equivalence class split by specializing on v, at least one of the resulting splits is of size less than k. Note that such values subsume the set of previously defined useless values. It can be shown that for both CM and DM metrics, any member of this more liberally defined set of values cannot possibly improve the cost of any descendant anonymization and can be pruned.

5.5 Tail Value Rearrangement

Recall that the tail set ordering affects the specific tree that will be enumerated. Before expanding the children of a given node H, our algorithm reorders its tail values in a manner that vastly increases pruning opportunities. While reordering strategies can be tailored to the specific cost metric used, we found one generic strategy to work well for both of our sample metrics. Given a node H, for each tail value v, the strategy counts the number of equivalence classes induced by H that are split by specializing on v. Tail values are sorted in decreasing order of this metric. Any ties are then broken by the smaller value of $\sum |E|^2$ over all equivalence classes E induced by the anonymization $H \cup v$.

This reordering strategy works well because specializations which have few positive effects stay in the tail sets the longest. If a node's tail consists of many such unproductive values, the lower-bounding techniques can often eliminate the node from consideration.

The impact of value rearrangement should not be underestimated. Without a good rearrangement strategy, good anonymizations will be scattered uniformly throughout the search tree, and it becomes impossible to prune significant portions of the search space.

5.6 Putting it All Together.

Pseudo-code for the K-OPTIMIZE algorithm is provided in Figure 5. While the procedure computes only the cost of an optimal anonymization, it can be trivially extended to also return the optimal anonymization itself. For clarity, we do not pass the dataset as an argument, and treat it as implicit.

The algorithm is initially invoked as K-OPTIMIZE(k, $\varnothing$, Σ_{all}, ∞) if no upper-bound on the best cost is known. In some cases, an upper-bound might be known from the cost of an anonymization determined by another algorithm, or from a previous run of K-OPTIMIZE with a higher value of k. In such a case, this cost can be provided in place of ∞ for better performance.

The main loop of K-OPTIMIZE implements a depth-first traversal of the set-enumeration tree. Note that it attempts to re-prune the tail after each recursive call; when returning from a recursive call, the algorithm has determined the best cost of any subnode that contains the value v used to extend H for that call. It can thus remove v from the tail to reduce the size of the allset. Because the lower-bound computations are dependent on allset equivalence classes, a reduced allset may allow additional pruning opportunities. This logic is also why The PRUNE function recursively calls itself if it is able to successfully prune any tail value. We omit pseudo-code for PRUNE-USELESS-VALUES, REORDER-TAIL, COMPUTE-COST, and COMPUTE-LOWER-BOUND

since their high-level descriptions follow directly from the previous discussion. The next section details data structures we have developed to support efficient execution of these operations.

6. Data Structures

6.1 Categories of Induced Equivalence Classes

A challenging aspect of implementing K-OPTIMIZE is in efficient cost and bound computation. As we have defined the algorithm, we require the three following categories of induced equivalence classes for computing costs and their bounds at a given node H:

(**Category 1**) the equivalence classes induced by the head anonymization H,

(**Category 2**) the equivalence classes induced by the anonymization $H \cup v$ for each tail value v, and

(**Category 3**) the equivalence classes induced by the allset anonymization.

Head equivalence classes are required for lower-bounding and for determining the cost of the current node. The next category of equivalence classes is required for tail value pruning and reordering. The allset equivalence classes are required for cost lower-bounding. Even though we have focused on two specific metrics, we note that this information would be required for computing and bounding most others.

6.2 Incremental Maintenance of Head Classes

A simple though slow approach to identifying equivalence classes induced by an anonymization is to first sort the dataset according to an equality function that compares the tuples in their transformed state. Then, equivalence classes can be demarcated by simply scanning the dataset to detect class boundaries. Rather than resorting to multiple dataset sorts per node as such a simple approach would imply, our implementation incrementally maintains the equivalence classes induced by the head set anonymization.

A child node is expanded by adding a tail value to the head set. Recall from Section 5.3 that the effect of such additional specialization is to split a subset of the existing induced equivalence classes into two, as depicted in Figure 4. Instead of sorting the dataset to identify the newly induced equivalence classes after expanding a child, our implementation identifies the relevant equivalence classes from the parent node and splits them according to the interval imposed by the new tail value.

In addition to supporting child expansion, the incremental structure needs to be able to recover the state of equivalence classes when backtracking from a child to its parent. To facilitate rapid recovery of the parent node's state, each time the algorithm splits an equivalence class, a pair of pointers to the newly split equivalence classes is placed on a stack. When time comes to backtrack, these pairs are popped off the stack, and the equivalence classes designated by each pair are merged back into a single class. This stack-based recovery is more efficient than the alternative of explicitly scanning equivalence classes in order to identify which ones must be merged to recover the parent state.

So maintained, this structure provides the set of category 1 equivalence classes induced by the current node at any point during the search.

6.3 Obtaining Tail and Allset-Induced Classes

Category 2 equivalence classes must be computed for each tail value v. In our implementation, this involves specializing the classes induced by the head anonymization for each tail value. We could perform the specialization by explicitly splitting the head equivalence classes in the usual way, followed by a recovery phase that immediately merges them before going on to the next tail value. Instead, the algorithm determines the resulting sizes of each split for each tail value without explicitly performing it.

Obtaining the category 3 allset equivalence classes is the most problematic since the allset typically contains many values. If the number of tail values is small, our implementation will apply the previous incremental specialization procedure repeatedly (one for each tail value) to determine the equivalence classes induced by the allset, followed by an immediate stack-based recovery. If there are many values in the tail set (> 20), then we have found it is more efficient to determine the allset equivalence classes by individually sorting and demarcating the already materialized equivalence classes induced by the head set. Since each equivalence class is generally much smaller than the entire dataset, each equivalence class sort exhibits significantly better locality than a complete dataset sort, yielding better performance. Correctness of this sorting optimization follows from Observation 5.2.

7. Evaluation

7.1 Experimental Setup

One goal of our experiments is to understand the performance of K-OPTIMIZE. Beyond this, the ability to quickly identify optimal solutions allows exploring many other interesting aspects of k-anonymization. For instance, we have noted that several variations on the problem have been proposed, including whether to allow (a bounded number of) suppressions, or whether to allow the algorithm to partition ordinal domains without hierarchy restrictions. Which variations are worthwhile in the sense that they produce "better" solutions? What is the impact of such extensions on performance? Another goal of the experiments is to begin addressing such concerns.

The dataset used in our experiments was the adult census dataset from the Irvine machine learning repository, since this dataset is the closest to a common k-anonymization "benchmark" that we are aware of. This dataset was prepared as described by Iyengar [5] to the best of our ability. It consists of 9 attributes (8 regular and one class column) and 30,162 records, and contains actual census data that has not already been anonymized. The so-coded dataset ("adult_fine") supports partitioning of the age column at a very fine grain (one value for each unique age

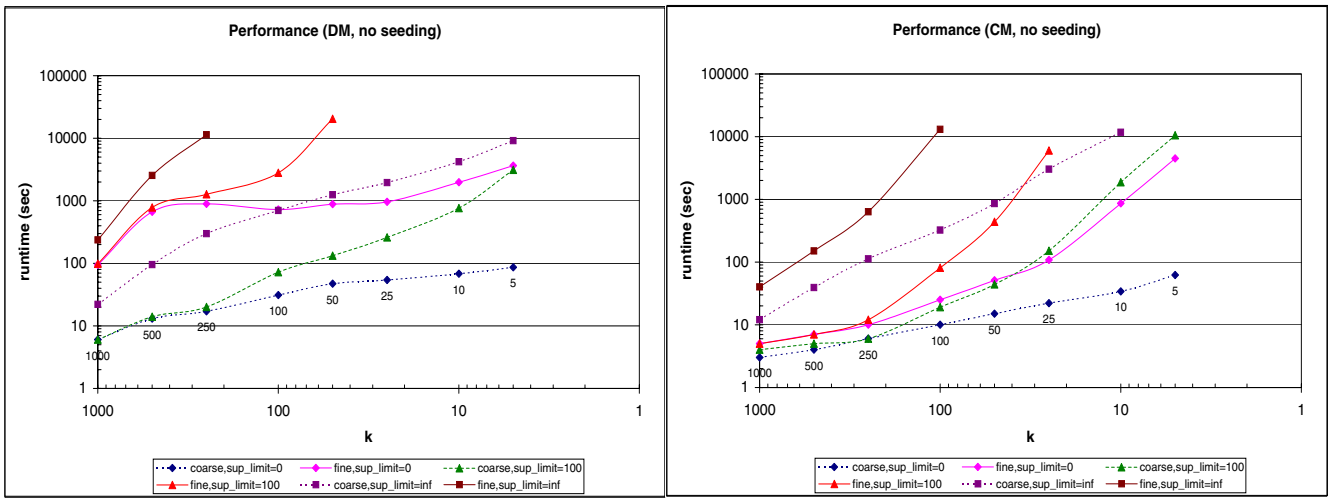

Figure 6a. Performance of the K-OPTIMIZE procedure.

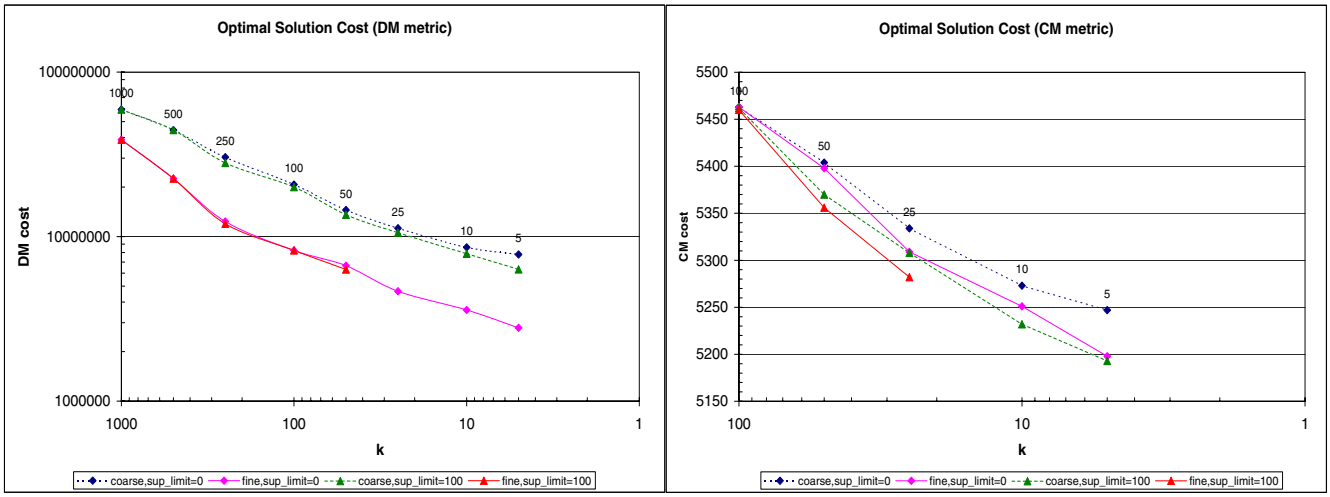

Figure 6b. Optimal solution cost.

rounded to the nearest year). This resulted in a searchable alphabet (Σ_{all}) of over 160 values.

To understand the performance and cost impacts of using a coarser initial partitioning, we also experimented with another coding of the dataset ("adult_coarse") in which the age column was "pre-generalized" into 15 unique values, each consisting of a 5 year age range.[1] This reduced the alphabet to a more manageable but still challenging 100 unique values. (Recall that this results in a state space of size 2^{100}.) Coarser partitionings reduce the flexibility given to the algorithm in determining an anonymization.

The algorithm was implemented in C++. All run-times are from a dedicated 2 processor, 2.8 GHZ Intel Xeon class machine running Linux OS (kernel version 2.4.20) and gcc v3.2.1. Only one processor was used by the algorithm during each run. We used the qsort C library function provided by gcc for all sorting operations.

[1] Though we used this simple equidistant pre-partitioning strategy for all experiments, a domain discretization approach that considers the class column [4] might be a better approach when using class-conscious metrics such as the classification metric.

7.2 Results

We explored the impacts of the following dimensions on performance and solution quality:

- The setting of k. Specifically, we used values of 1000, 500, 250, 100, 50, 25, 10, and 5.
- The number of suppressions allowed. Specifically, we ran the algorithm with no suppressions allowed, a limit of 100 suppressions, and with no restriction (denoted *inf*) on the number of suppressions.
- The impact of seeding the algorithm with a non-infinite cost.
- The impact of a coarse versus fine partitioning of the age column.
- The cost metric used (CM or DM.)

The first graphs (Figure 6a) show the runtime when there is no cost seeding. Note that the algorithm performs

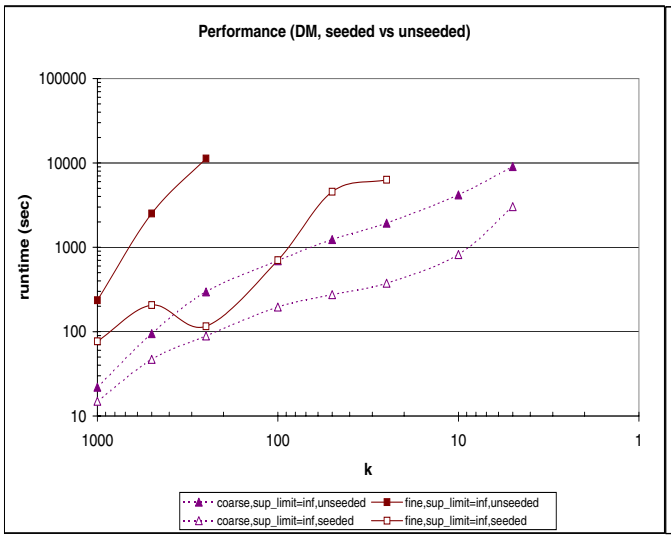

Figure 7a. Seeded vs. Unseeded Performance of K-OPTIMIZE.

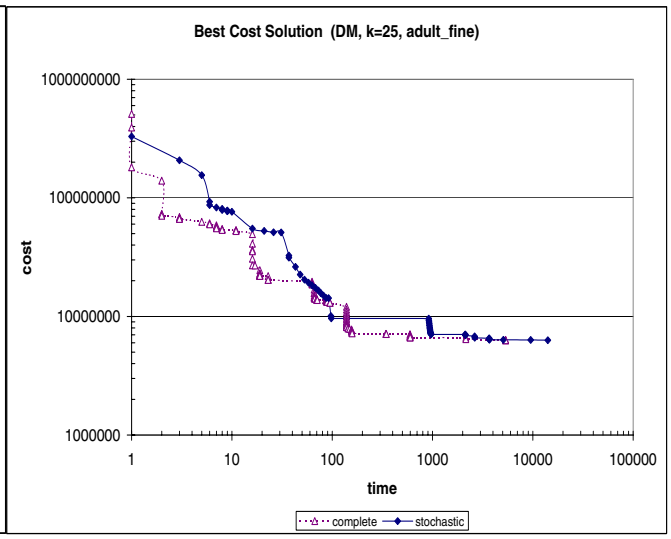

Figure 7b. Stochastic and K-OPTIMIZE solution cost over time.

extremely well across all values of k when no suppressions are allowed. As we allow more suppressions, algorithm performance degrades rapidly with smaller k. As expected, the vastly increased search space resulting from fine partitioning of the age column requires significantly more time, though the algorithm still successfully identifies the optimal anonymization for all but a very few cases. We noticed that the higher the limit on suppressions, the more slowly the algorithm converged on a good solution. This suggested to us that seeding the algorithm with a good initial score might substantially improve its overall performance when the suppression limit was high. We verified this was indeed the case (Figure 7b) by seeding the algorithm with the optimal cost identified by the no suppression runs. The runtimes plotted in this figure add the cost required to find the no suppression optimal solution with the cost required to find the optimal solution after seeding. More performance improvements might be possible if we were to seed with the best solution score identified by the suppression-free run after some fixed amount of time, since the suppression-free algorithm typically identified good (though not necessarily optimal) solutions very quickly.

Regarding cost of the optimal solutions (Figure 6b), we found that the finely partitioned age column allowed for significant improvements in cost for both metrics. Interestingly, for all cases, allowing an infinite number of suppressions did not improve the cost of the optimal solution at all, which is why the sup_limit=inf plots are omitted in these graphs. The coarsely partitioned dataset benefited more from allowing suppressions, but significant benefits from suppressions appear only at smaller settings of k. For the CM metric, we plot only the results for $k \leq 100$, since optimal solution cost for higher settings of k was identical across all other parameter combinations.

7.3 Comparison with Other Algorithms

While none of the other optimal algorithms in the literature can feasibly handle the census dataset used in our experiments, incomplete methods are known to perform well in practice on this data despite their lack of any quality guarantees. We therefore compared the effectiveness of K-OPTIMIZE to various incomplete methods. These methods included simulated annealing, genetic algorithms, and greedy (hill-climbing) algorithms. We found that purely greedy approaches, though they executed quite quickly, typically produced highly sub-optimal anonymizations. Genetic algorithms and simulated annealing produced anonymizations of higher quality, but converged too slowly. The approach we found to work best was a new iterated 2-phase hill-climbing method we concocted to combine the virtues of both greedy and stochastic methods. This algorithm begins by generating a random anonymization. It then enters a generalization phase in which values are iteratively removed (always selecting the one that most improves the cost) until the cost can no longer be improved. Next, it switches to a specialization phase in which values are iteratively added (again always selecting the one providing the biggest cost improvement) until no improvement is possible. The algorithm repeatedly executes these two phases until neither phase is capable of improving the score (implying a local minimum is reached.) At this point, the anonymization cost is recorded, and the algorithm repeats the entire process. This algorithm has no explicit stopping criteria and instead continuously attempts to improve on the best solution found until stopped by the user.

Note that the specialization phase of this algorithm can be optimized according to the strategies provided in Section 6, and similar approaches applied for optimizing the

generalization phase. However, the implementation evaluated here uses only dataset sorting for cost computations.

When left to run long enough, this new stochastic approach is typically capable of finding an optimal anonymization even though it is of course incapable of guaranteeing its optimality. The graph in Figure 7b depicts the cost of the best solution found by both this stochastic approach and K-OPTIMIZE (labeled "complete") at a given point during their execution. We ran K-OPTIMIZE with a zero suppression limit followed by a seeded run with no suppression limit. Likewise, the stochastic approach was allowed to consider solutions with no limit on the number of suppressions. Note that while K-OPTIMIZE was not designed for rapidly finding a good solution, it performs similarly to the stochastic method for this purpose, and finds the optimal solution three times more quickly (5359 vs. 14104 seconds.) The figure depicts the case where $k = 50$ on the finely partitioned dataset; results for other parameter settings were qualitatively similar. We believe an improved implementation of this new stochastic approach that reduces reliance on sorting operations will prove attractive in practice when provable optimality is either out of reach or not a requirement.

8. Conclusions and Future Work

We have proposed an algorithm that identifies provably optimal anonymizations of real census data under a flexible and highly combinatorial model of attribute value generalization. We framed the problem as one that involves searching the power-set of a special alphabet of domain values, and attacked it through a tree-search strategy that explores anonymizations beginning from the most general to more specific. The algorithm incorporates node pruning through cost lower-bounding and dynamic tree rearrangement through tail value reordering. In addition, we implemented data-management strategies that avoid repeatedly sorting the entire dataset for markedly reduced node evaluation times. We used the algorithm to quantify the effects of various parameter settings and data preparation methods on performance as well as anonymization quality. We also used the approach to evaluate effectiveness of stochastic methods for k-anonymization, and proposed a new iterated 2-phase greedy algorithm that outperforms other incomplete methods.

A desirable feature of protecting privacy through k-anonymity is its preservation of data integrity. Despite its intuitive appeal, it is possible that non-integrity preserving approaches to privacy (such as random perturbation) may produce a more informative result in many circumstances. Indeed, it may be interesting to consider combined approaches, such as k-anonymizing over only a subset of potentially identifying columns and randomly perturbing the others. We believe that a better understanding of when and how to apply various privacy-preserving methods deserves further study. Optimal algorithms will be useful in this regard since they eliminate the possibility that a poor outcome is the result of a highly sub-optimal solution rather than an inherent limitation of the specific technique.

Acknowledgements

We thank Vijay Iyengar for his valuable feedback and his assistance with several data preparation issues.

References

[1] C. C. Aggarwal and P. S. Yu. A condensation approach to privacy preserving data mining. *Advances in Database Technology - EDBT-2004, 9th Int'l Conf. on Extending Database Technology*, 183-199.

[2] R. Agrawal and R. Srikant. Privacy preserving data-mining. In *Proc. of the ACM SIGMOD Conf. on Management of Data* 439-450, 2000.

[3] R. J. Bayardo, R. Agrawal, and G. Gunopulos. Constraint-based rule mining in large, dense databases. In *Proc. of the 15th Int'l Conf. on Data Engineering,* 188-197, 1999.

[4] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. of 13th Int'l Joint Conf. on Artificial Intelligence*, 1022-1027, 1993.

[5] V. Iyengar. Transforming data to satisfy privacy constraints. In *Proc. of the Eigth ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, 279-288, 2002.

[6] A. Hundpool and L. Willenborg. Mu-Argus and Tau Argus: Software for Statistical Disclosure Control. *Third Int'l Seminar on Statistical Confidentiality*, 1996.

[7] S. Kim and W. Winkler. Masking microdata files. In *ASA Proc. of the Section on Survey Research Methods*, 114-119, 1995.

[8] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *Proc.of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on the Principles of Database Systems*, 223-228, 2004.

[9] R. Rymon. Search through systematic set enumeration. In *Proc. of the Third Int'l Conf. on Principles of Knowledge Representation and Reasoning*, 539-550, 1992.

[10] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering* **13**(6): 1010-1027, 2001.

[11] P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information. In *Proc. of the 17th ACM SIGMOD-SIGACT-SIGART Symposium on the Principles of Database Systems*, 188, 1998.

[12] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *Int'l Journal on Uncertainty, Fuzziness, and Knowledge-Base Systems* **10**(5): 571-588, 2002.

[13] L. Sweeney. Datafly: a system for providing anonymity in medical data. In *Database Security XI: Status and Prospects, IFIP TC11 WG11.3 11th Int'l Conf. on Database Security*, 356-381, 1998.

[14] L. Sweeney. K-anonymity: a model for protecting privacy. *Int'l Journal on Uncertainty, Fuzziness, and Knowledge-Based Systems* **10**(5):557-570, 2002.

[15] G. I. Webb. Opus: an efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research* **3**, 431-465, 1995.

[16] W. E. Winkler. *Using Simulated Annealing for k-anonymity.* Research Report Series (Statistics #2002-7), U. S. Census Bureau, 2002.