Scalable Anonymization Algorithms for Large Data Sets

Kristen LeFevre and David DeWitt
University of Wisconsin – Madison
Department of Computer Sciences
Technical Report 1590
March 1, 2007

ABSTRACT

k-Anonymity is a widely-studied mechanism for protecting identity when distributing non-aggregate personal data. This basic mechanism can also be extended to protect an individual-level sensitive attribute. Numerous algorithms have been developed in recent years for generalizing, clustering, or otherwise manipulating data to satisfy one or more anonymity requirements. However, few have considered large-scale input data sets that do not fit in main memory.

This paper proposes two techniques for incorporating (external) scalability into an existing algorithmic framework. The first technique is based on ideas from scalable decision tree construction, and the second technique is based on sampling. In both cases, the resulting algorithms are guaranteed to produce output data that satisfies the given anonymity requirements. We evaluate the performance of each algorithm both analytically and experimentally.

1. INTRODUCTION

Many organizations distribute non-aggregate, individual-level, data for demographic and medical research. Unfortunately, published data sets can often be "linked" with other publicly-available data to re-identify individuals (and their sensitive attributes). Recently, the k-anonymity model [19, 21], as well as a variety of extensions, have drawn a lot of attention as mechanisms for limiting the risk of this kind of attack.

A wide variety of algorithms have been proposed for generalizing, clustering, or aggregating data to guarantee k-anonymity [1, 2, 3, 5, 6, 7, 10, 12, 16, 19, 20, 22, 24]. However, the vast majority of proposals assume that the data fits in main memory. For larger data sets, memory and I/O must be managed explicitly to present thrashing and performance degradation.

In this paper, we propose two new techniques for scaling an existing generalization framework, Mondrian [13, 14], to data sets much larger than the available memory.

1.1 Preliminaries

This paper considers the well-known problem of using generalization to publish a single view (R^*) of a single base relation (R), while limiting the risk of a linking attack. We assume, as in the majority of previous work, that each attribute in R can be uniquely characterized by at most one of the following types based on knowledge of the application domain:

- **Identifier** Unique identifiers, such as *Social Security Number*, are removed entirely from the published data.
- **Quasi-Identifier** The quasi-identifier is a set of attributes $Q_1, ..., Q_d$ that can potentially be used to re-identify individuals when combined with other public data, for example, *Age*, *Sex*, and *Zipcode*.
- **Sensitive Attribute** An attribute, such as *Disease*, is considered sensitive if an adversary should not be permitted to uniquely associate its value with a unique identifier.

k-Anonymity provides a simple and intuitive means for protecting individual *identity* with respect to linking attacks [19, 21]. It stipulates that no individual record should be uniquely identifiable from a group of less than k on the basis of its quasi-identifier values. (Throughout the paper, we assume bag semantics.) We will refer to each group of tuples in R^* with identical quasi-identifier values as an *equivalence class*.

k-Anonymity R^* is *k-anonymous* with respect to quasi-identifier attributes $Q_1, ..., Q_d$ if every unique tuple $(q_1, ..., q_d)$ in the projection of R^* on $Q_1, ..., Q_d$ occurs at least k times.

Machanavajjhala et al. noted that it is often natural to extend the k-anonymity model to protect a known *sensitive attribute* S [15]. When S is categorical, this idea can be implemented in several ways. Let $\texttt{dom}(S)$ denote the domain of attribute S in R.

Recursive (c, ℓ)-Diversity Within a given equivalence class (E), let x_i denote the number of times the i^{th} most frequent sensitive value appears in E. Given a constant c, E satisfies recursive (c, ℓ)-diversity if $x_1 < c(x_\ell + x_{\ell+1} + ... + x_{|\texttt{dom}(S)|})$. Table R^* satisfies recursive (c, ℓ)-diversity if each equivalence class satisfies recursive diversity. (Define (c, ℓ)-diversity to be always satisfied.)

For numeric S, the same intuition can be extended to require a minimum level of dispersion of S within each equivalence class [14], and we use a variant of this proposal.[1] Let $Var(E, S) = \frac{1}{|E|}\sum_{i=1}^{m}(s_i - \overline{s})^2$ denote the variance of values for sensitive attribute S among tuples in equivalence class E. ($\overline{s}$ denotes the mean value of S in E.)

Variance Diversity R^* is variance diverse with respect to sensitive attribute S if, for each equivalence class E in R^*, $Var(E, S) \geq v$, where v is the diversity parameter.

1.2 Greedy Partitioning Algorithm

In previous work, we proposed implementing these anonymity requirements using a partitioning approach [13, 14]. The idea is to divide the d-dimensional quasi-identifier domain space into non-overlapping rectangular regions. This partitioning is used to define a *global recoding function* ($\phi : \texttt{dom}(Q_1) \times .. \times \texttt{dom}(Q_d) \rightarrow D^d$) that maps each domain tuple to the region in which it is contained.[2]

[1] Variance diversity also satisfies the monotonicity property, as described in [15]; see Appendix for proof.

[2] Each d-dimensional region, in turn, can be represented as a tuple in tabular form through the use of generalization, range values, summary statistics, etc.

Figure 1: Generalization hierarchy for Nationality

A partitioning is allowable with respect to a particular input relation R if the "recoded" relation R^*, resulting from applying ϕ to the quasi-identifier attributes of R, satisfies all given anonymity requirements.

The proposed algorithm (*Mondrian*) is based on greedy recursive partitioning [13]. Briefly, the recursive procedure takes as input a (potentially infinite) d-dimensional rectangular domain, and a set of tuples, R. The algorithm chooses a quasi-identifier *split attribute* (dimension of the domain space). When the split attribute is numeric, the algorithm also chooses a binary split *threshold* (e.g., $Age \leq 40$; $Age > 40$). For categorical attributes, the split is defined by specializing a user-defined generalization hierarchy (e.g., Figure 1), as originally proposed by Samarati and Sweeney [19, 20]. We use the notation $\preceq$ to indicate a generalization relationship.

The split attribute (and threshold) define a division of the input domain into m non-overlapping regions that cover the input domain. The split also defines a corresponding partitioning of the input data (R) into disjoint subsets, $R_1, .., R_m$. The split is said to be ***allowable*** if each R_i satisfies the given anonymity requirement(s). For example, under k-anonymity, a split is allowable if each R_i contains at least k tuples.

The procedure is executed recursively on each resulting partition (R_i), until there no longer exists an allowable split. Informally, a partitioning is said to be ***minimal*** if it satisfies the given anonymity requirement(s), and there exist no further allowable splits.

Motivated by various target workloads, including classification and regression, the recursive procedure can use one of several heuristics to choose the split attribute (and threshold):

- **Median** When there is no known target workload, [13] proposed choosing the allowable split attribute with the widest (normalized) range of values, and (for numeric attributes) used the median value as the threshold. Median partitioning is appealing because (under k-anonymity) if there exists an allowable split perpendicular to a particular axis, the split at the median is necessarily allowable, and can be found in linear time.

- **InfoGain** For classification tasks, [14] proposed choosing the allowable split resulting in maximum information gain with respect to a categorical class label. For numeric attributes, the best allowable threshold is chosen by sorting the data on the candidate split attribute, and then evaluating candidate thresholds. [3]

- **Regression** For regression tasks, [14] suggested choosing the allowable split minimizing the sum of squared error with respect to a numeric target attribute. Again, for numeric attributes, the best allowable threshold is chosen by first sorting the data.

Under k-anonymity, for constant dimensionality (d), the complexity of the Median partitioning algorithm is $O(|R| \log |R|)$. For numeric attributes, all other cases (InfoGain or Regression splitting, entropy ℓ-diversity, and variance diversity) require additional sorting in order to choose the best allowable thresholds. For this reason, the complexity of these algorithms is $O(|R| \log^2 |R|)$.

[3]For classification and regression tasks, we assume that the class label or numeric target attribute, respectively, is not also sensitive.

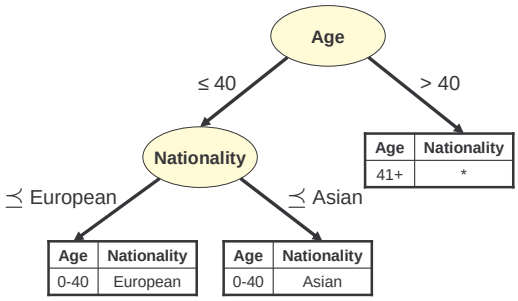


Figure 2: Partition tree defining a global recoding function

Following partitioning, the global recoding function ϕ can be defined by the resulting *partition tree*. For example, Figure 2 shows a partition tree for quasi-identifier attributes Age and Nationality. Notice that the generalized tuples (located in the leaves) cover the entire domain space.

Alternatively, each leaf can be represented using various statistics summarizing the set of data tuples contained therein (e.g., mean, minimum, maximum, etc.) [3, 13], a technique often used in the related problem of microaggregation [6].

1.3 Paper Outline

The main contribution of this paper is a pair of simple and effective algorithmic variations, each of which allows the Mondrian framework to be applied to data sets larger than main memory. For clarity, we refer to the scalable variations as *Rothko*.[4]

Our first scalable algorithm, described in Section 2, is based on ideas from the RainForest scalable decision tree algorithms [9]. Two main challenges had to be addressed in order to adapt the RainForest framework to scalable anonymization. First, in order to choose an allowable split (according to a given split criterion and anonymity requirement), we need to choose an appropriate set of count statistics; those used in RainForest are not always sufficient. Also, we note that in the anonymization problem, as opposed to the decision tree problem, the resulting partition tree does not necessarily fit in memory, and we propose techniques addressing this problem.

The second algorithm (Section 3) takes a different approach, based on sampling. The main idea is to use a sample of the input data set R (that fits in memory), and to build the partition tree optimistically according to the sample. Any (non-allowable) split made in error is subsequently undone; thus, the output is guaranteed to satisfy all given anonymity requirements. We find that, for non-trivial sample sizes, this algorithm almost always produces a minimal partitioning.

Finally, we evaluate the proposed techniques using both a back-of-the-envelope analysis of I/O behavior (Section 4) and an extensive experimental study (Section 5). We find that, when applied naively to large data sets, the in-memory algorithms described in [13, 14] often lead to thrashing, and correspondingly poor performance. However, both of the scalability techniques described in

[4]Mark Rothko (1903-1970) was a Latvian-American painter, whose late abstract expressionist work was influenced by Piet Mondrian, among others.

this paper substantially improve performance, and the sampling-based approach is often fastest.

2. EXHAUSTIVE ALGORITHM (ROTHKO-T)

Our first algorithm, which we call Rothko-Tree (or Rothko-T), leverages several ideas originally proposed as part of the RainForest scalable decision tree framework [9]. Like Mondrian, decision tree construction typically involves a greedy recursive partitioning of the domain (feature) space. For decision trees, Gehrke et al. observed that split attributes (and thresholds) could be chosen using a set of count statistics, typically much smaller than the full input data set [9].

In many cases, allowable splits can be chosen greedily in Mondrian using related count statistics, each of which is typically much smaller than the size of the input data.

- **Median / k-Anonymity** Under k-anonymity and Median partitioning, the split attribute (and threshold) can be chosen using what we will can an *AV group*. The *AV set* of attribute A for tuple set R is the set of unique values of A in R, each paired with an integer indicating the number of times it appears in R (i.e., SELECT A, COUNT(*) FROM R GROUP BY A). The AV group is the collection of AV sets, one per quasi-identifier attribute.

- **InfoGain / k-Anonymity** When the split criterion is InfoGain, each AV set (group) must be additionally augmented with the class label, producing an *AVC set (group)*, as described in [9] (i.e., SELECT A, C, COUNT(*) FROM R GROUP BY A, C.).

- **Median / ℓ-Diversity** In order to determine whether a candidate split is allowable under ℓ-diversity, we need to know the joint distribution of attribute values and sensitive values, for each candidate split attribute (i.e., SELECT A, S, COUNT(*) FROM R GROUP BY A, S). We call this the *AVS set (group)*.

- **InfoGain / ℓ-Diversity** Finally, when the split criterion is InfoGain, and the anonymity constraint is ℓ-diversity, the allowable split yielding maximum information gain can be chosen using *both* the AVC and AVS groups.

Throughout the rest of the paper, when the anonymity requirement and split criterion are clear from context, we will interchangeably refer to the above as *frequency sets* and *frequency groups*.

When the anonymity constraint is variance diversity, or the split criterion is Regression, the analogous summary counts (e.g., the joint distribution of attribute A and a numeric sensitive attribute S, or numeric target attribute T) are likely to be prohibitively large. We return to this issue in Section 3.

In the remainder of this section, we describe a scalable algorithm for k-anonymity and/or ℓ-diversity (using Median or InfoGain splitting) based on these summary counts. In each case, the output of the scalable algorithm is identical to the output of the corresponding in-memory algorithm [13, 14].

2.1 Algorithm Overview

The recursive structure of Rothko-T follows that of RainForest [9], and we assume that at least one frequency group will fit in memory. In the simplest case, the algorithm begins at the root of the partition tree, and scans the input data (R) once to construct the frequency group. Using this, it chooses an allowable split attribute (and threshold), according to the given split criterion. Then, it scans R once more, and writes each tuple to a disk-resident child partition, as designated by the chosen split. The algorithm proceeds recursively, in a depth-first manner, dividing each of the resulting partitions (R_i) according to the same procedure.

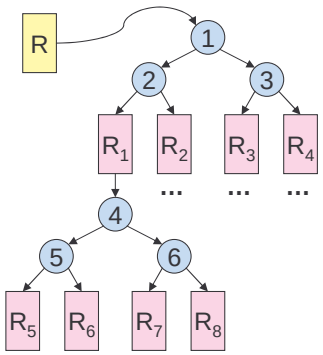


Figure 3: Example for Rothko-T

Once the algorithm descends far enough into the partition tree, it will reach a point where the data in each leaf partition is small enough to fit in memory. At this point, a sensible implementation loads each partition (individually) into memory, and continues to apply the recursive procedure in memory.

When multiple frequency groups fit in memory, the simple algorithm can be improved to take better advantage of the available memory, using an approach reminiscent of the RainForest hybrid algorithm. In this case, the algorithm first scans R, choosing the split attribute and threshold using the resulting frequency group. Now, suppose that there is enough memory available to (simultaneously) hold the frequency groups for all child partitions. Rather than repartitioning the data across the children, the algorithm proceeds in a breadth-first manner, scanning R once again to create frequency groups for all of the children.

Because the number of partitions grows exponentially as the algorithm descends in the tree, it will likely reach a level at which all frequency groups no longer fit in memory. At this point, it divides the tuples in R across the leaves, writing these partitions to disk. The algorithm then proceeds by calling the procedure recursively on each of the resulting partitions.[5] Again, when each leaf partition fits in memory, a sensible implementation switches to the in-memory algorithm.

Example (Rothko-T) Consider input tuple set (R), and suppose there is enough memory available to hold 2 frequency groups for R. The initial execution of the algorithm is depicted in Figure 3.

Initially, the algorithm scans R once to create the frequency group for the root (1) and chooses the best allowable split (provided that one exists). (In this example, all of the splits are binary.) Then, the algorithm scans R once more to construct the frequency groups for the child nodes (2 and 3), and chooses the best allowable splits for these nodes.

Following this, the four frequency groups for the next level of the tree will not fit in memory, so the data is divided into partitions $R_1, ..., R_4$. The procedure is then called recursively on each of the resulting partitions.

2.2 Recoding Function Scalability

Because the decision trees considered by Gehrke et al. were of approximately constant size, it was reasonable to assume that the

[5]Note that RainForest proposed additionally caching a set of frequency groups when repartitioning the data [9]. We found that with modern memory sizes, where many frequency groups fit in memory, the effects of this additional optimization were small.

resulting tree structure itself would fit in memory [9]. Unfortunately, this is often not true of our problem.

Instead, we implemented a simple scalable technique for materializing the multidimensional recoding function ϕ. Notice that each path from root to leaf in the partition tree defines a rule, and the set of all such rules defines global recoding function ϕ. For example, in Figure 2, $(Age < 40) \wedge (Nationality \preceq European) \rightarrow \langle [0-40], European \rangle$ is one such rule.

The set of recoding rules can be constructed in a scalable way, without fully materializing the tree. In the simplest case, when only one frequency group fits in memory, the algorithm works in a purely depth-first manner. At the end of each depth-first branch, we write the corresponding rule (the path from root to leaf) to disk. This simple technique guarantees that the amount of information stored in memory at any one time is proportional to the height of the tree, which grows only as a logarithmic function of the data.

When more memory is available for caching frequency groups, the amount of space is slightly larger, due to the periods of breadth-first partitioning, but the approach still consumes much less space than materializing the entire tree in memory.

Finally, note that the tree structure is only necessary if it is used to define a global recoding function that covers the domain space. If we instead choose to represent each resulting region using summary statistics, then the tree structure need not be materialized. Instead, the summary statistics can be computed directly from the resulting data partitions.

3. SAMPLING ALGORITHM (ROTHKO-S)

In this section, we describe a second scalable algorithm, this time based on sampling. Rothko-Sampling (or Rothko-S) addresses some of the shortcomings of Rothko-T. Specifically, because splits are chosen using only memory-resident data, it provides mechanisms both for choosing split attributes using the Regression split criterion, and for checking variance diversity. The sampling approach also often leads to better performance.

The main recursive procedure consists of three phases:

1. **(Optimistic) Growth Phase** The procedure begins by scanning input tuple set R to obtain a simple random sample (r) that fits in the available memory. (If R fits in memory, then $r = R$.) The procedure then grows the tree, using sample r to choose split attributes (thresholds). When evaluating a candidate split, it uses the sample to estimate certain characteristics of R, and using these estimates, it will make a split (optimistically) if it can determine with high confidence that the split will not violate the anonymity requirement(s) when applied to the full partition R. The specifics of these tests are described in Section 3.1.

2. **Repartitioning Phase** Eventually, there will be no more splits that can be made with high confidence based on sample r. If $r \subset R$, then input tuple set R is divided across the leaves of the tree built during the growth phase.

3. **Pruning Phase** When $r \subset R$, there is the possibility that certain splits were made in error during the growth phase. Given a reasonable testing procedure, this won't happen often, but when a node in the partition tree is found to violate (one of) the anonymity requirement(s), then all of the partitions in the subtree rooted at the parent of this node are merged. To do this, during the repartitioning phase, we maintain certain population statistics at each node. (For k-anonymity, this is just a single integer count. For ℓ-diversity or variance diversity, we construct a frequency histogram over the set of unique sensitive values.)

Finally, the procedure is executed recursively on each resulting partition, $R_1, ..., R_m$. In virtually all cases, the algorithm will eventually reach a base case where each recursive partition R_i fits entirely in memory. (There are a few pathological exceptions, which we describe in Section 3.2. These cases typically only arise when an unrealistically small amount of memory is available.)

Recoding function scalability can be implemented as described in Section 2.2. In certain cases, we stop the growth phase early, for one of three possible reasons. First, if we are constructing a global recoding function, and the tree structure has filled the available memory, we then write the appropriate recoding rules to disk. Similarly, we repartition the data if the statistics necessary for pruning (e.g., sensitive frequency histograms) no longer fit in memory. Finally, notice that repartitioning across a large number of leaves may lead to a substantial amount of nonsequential I/O if there is not enough memory available to adequately buffer writes. In order to prevent this from occurring, the algorithm may repartition the data while there still exist high-confidence allowable splits.

Example (Rothko-S) Consider an input tuple set R. The algorithm is depicted in Figure 4.

The growth phase begins by choosing sample r from R, and growing the partition tree accordingly. When there are no more (high-confidence) allowable splits, R is repartitioned across the leaves of the tree (e.g., Figure 4(a)).

During repartitioning, the algorithm tracks necessary population statistics for each node (e.g., total count for k-anonymity). In the example, suppose that Node 7 violates the anonymity requirement (e.g., contains fewer than k tuples). In this case, the tree is pruned, and partitions R_6, R_7, R_8 combined.

Finally, the procedure is executed recursively on data partitions $R_1, ..., R_5, R_6 \cup R_7 \cup R_8$.

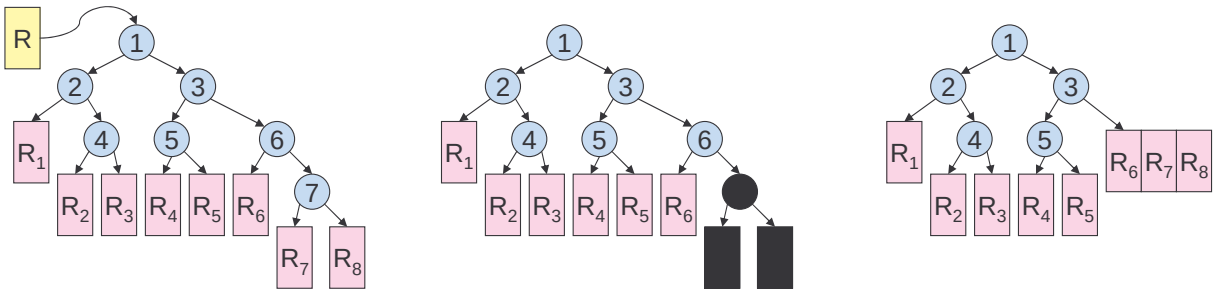
3.1 Estimators & Hypothesis Tests

Rothko-S must often use a sample to check whether a candidate recursive split satisfies the given anonymity requirement(s). A naive approach performs this check directly on the sample. For example, under k-anonymity, if input data R contains N tuples, and we have selected a sample of size n, the naive implementation makes a split (optimistically) if each resulting sample partition contains at least $k\left(\frac{n}{N}\right)$ tuples.

Unfortunately, we find that this naive approach can lead to an excessive amount of pruning in practice (Section 5.6). Instead, we propose to perform this check based on a statistical hypothesis test. In this section, we outline some preliminary methods for performing these tests. We find that, while our tests for variance diversity and ℓ-diversity do not make strong guarantees, these tests produce quite favorable results in practice. Similarly, the test does not affect the anonymity of the resulting data because the algorithm always undoes any split made in error.

In the context of splitting, the *null hypothesis* (H_0) can be described informally as stating that the candidate split is *not* allowable under the given anonymity requirement. An ideal test would reject H_0 if it can determine (using realistic assumptions) that there is only a small probability ($\leq \alpha$) of the split violating the anonymity requirement. During the growth phase, Rothko-S will make a split (optimistically) if H_0 can be rejected with high confidence.

In the following, let R denote the input data (a population of tuples), and let N denote the size (number of tuples) of R. Let r denote a simple random sample of n tuples, drawn uniformly without replacement from R ($n \leq N$). Consider a candidate split, which divides R into m partitions $R_1, ..., R_m$. (When applied to sample r, the split yields sample partitions $r_1, ..., r_m$.)



(a) (b) (c)

Figure 4: Example for Rothko-S

3.1.1 *k-Anonymity*

We begin with k-anonymity. Let $p = |R_i|/N$ denote the proportion of tuples from R that would fall in partition R_i after applying a candidate split to R. Under k-anonymity, H_0 and H_1 can be expressed (for R_i) as follows, where $p_0 = k/N$.

$$\begin{aligned} H_0 &: p = p_0 \\ H_1 &: p \geq p_0 \end{aligned}$$

Similarly, let $\hat{p} = |r_i|/n$. We use proportion $\hat{p}$ to estimate p. Regardless of the underlying data distribution, we know by the central limit theorem that $\hat{p}$ is approximately normally distributed (for large samples) [18]. Thus, we use the following test, rejecting H_0 when the expression is satisfied.[6]

$$\begin{aligned} Var_{H_0} &= \frac{p_0(1-p_0)}{n}\left(\frac{N-n}{N-1}\right) \\ \hat{p} - p_0 &\geq z_{\alpha/m}\sqrt{Var_{H_0}} \end{aligned}$$

There are three important things to note about this test. First, notice that we are simultaneously testing all m partitions resulting from the split. That is, we want to construct the test so that the total probability of accepting any R_i containing fewer than k tuples is α. For this reason we use the Bonferroni correction (α/m).

Also, it is important to remember that we are sampling from a finite population of data (R), and the fraction of the population that fits in memory (and is included in the sample) grows each time the algorithm repartitions the data. For this reason, we have defined Var_{H_0} in terms of the sampling process, incorporating a finite population correction. Given this correction, notice that when $N = n$ (i.e., the entire partition fits in memory), then $Var_{H_0} = 0$.

Finally, as the growth phase progresses (prior to repartitioning), note that the population (R), and the sample (r), do not change. The only component of the hypothesis test that changes during a particular instantiation of the growth phase is $\hat{p}$, which decreases with each split. Thus, as the growth phase progresses, it becomes increasingly likely that we will be unable to reject H_0.

3.1.2 *ℓ-Diversity*

When the anonymity requirement is recursive (c, ℓ)-diversity, we must use each sample partition (r_i) to estimate certain characteristics of the sensitive attribute S within the corresponding population partition (R_i). Let N_i denote the size of population partition R_i, and let n_i denote the size of sample partition r_i.

Recursive (c, ℓ)-diversity can be expressed in terms of two proportions. Let X_j denote the frequency of the j^{th} most common sensitive value in R_i. Let $p_1 = X_1/N_i$ and $p_2 = (X_\ell + ... + X_{|dom(S)|})/N_i$. Using these proportions,

$$\begin{aligned} H_0 &: p_1 = c * p_2 \\ H_1 &: p_1 < c * p_2 \end{aligned}$$

We use the sample partition (r_i) to estimate these proportions. Let x_j denote the frequency of the j^{th} most common sensitive value in r_i, and let $\widehat{p_1} = x_1/n_i$ and $\widehat{p_2} = (x_\ell + ... + x_{|dom(S)|})/n_i$.

Notice that these estimates make several implicit assumptions. First, they assume that the domain of sensitive attribute S is known. More importantly, they assume that the ordering of sensitive value frequencies is the same in R_i and r_i. In practice, this leads to a conservative bias for small sample sizes. Nonetheless, this seems to be a reasonable rule of thumb for larger samples, and a good starting point.[7]

In order to do the test, we need to estimate the sample variance of $c\widehat{p_2} - \widehat{p_1}$. If we assume that $\widehat{p_1}$ and $\widehat{p_2}$ are independent (which is not true), then

$$Var(c\widehat{p_2} - \widehat{p_1}) = c^2 Var(\widehat{p_2}) + Var(\widehat{p_1})$$

An estimator for $Var(\hat{p})$ is $\frac{\hat{p}(1-\hat{p})}{n_i - 1}\left(\frac{N_i - n_i}{N_i}\right)$, so we estimate the variance as follows.

$$\frac{c^2\widehat{p_2}(1-\widehat{p_2}) + \widehat{p_1}(1-\widehat{p_1})}{n_i - 1}\left(\frac{N_i - n_i}{N_i}\right)$$

Of course, when choosing a candidate split, we do not know N_i, the size of the i^{th} resulting population partition. Instead, we use the overall sampling proportion ($\frac{n}{N}$) to guide the finite population correction, which gives us the following estimate.

$$\widehat{Var}_{H_0} = \frac{c^2\widehat{p_2}(1-\widehat{p_2}) + \widehat{p_1}(1-\widehat{p_1})}{n_i - 1}\left(\frac{N - n}{N}\right)$$

Finally, we reject H_0 in favor of H_1 when the following expression is satisfied, again using the Bonferroni correction.

$$c\widehat{p_2} - \widehat{p_1} > z_{\alpha/m}\sqrt{\widehat{Var}_{H_0}}$$

3.1.3 *Variance Diversity*

When the anonymity requirement is variance diversity, we again use the sample partition r_i to estimate certain characteristics of R_i, namely the variance of sensitive attribute S.

The null and alternative hypotheses (for population partition R_i) can be expressed as follows.

$$\begin{aligned} H_0 &: Var(R_i, S) = v \\ H_1 &: Var(R_i, S) \geq v \end{aligned}$$

[6] z_α is the number such that the area beneath the standard normal curve to the right of $z_\alpha = \alpha$.

[7] We also considered an alternate definition, entropy ℓ-diversity [15], and found it equally difficult to develop a precise test without simulating H_0.

We use the sample variance of S within r_i to estimate the variance of the sensitive attribute in population partition R_i.

We use the variance of S within sample partition r_i as an estimate of the variance in population partition R_i.

$$\widehat{Var}(R_i, S) = \frac{1}{n_i - 1} \sum_{j=1}^{n_i} (s_j - \overline{s})^2$$

Recall that if each s_j is an independent normally-distributed random variable, then the sample variance of S follows a chi-square distribution. Thus, we reject H_0 (for R_i) if the following holds.[8]

$$\frac{(n_i - 1)\widehat{Var}(R_i, S)}{v} \geq \chi^2_{\alpha/m}\ (n_i - 1\ df)$$

In practice, this test may amount to little more than a rule of thumb because S may follow an arbitrary distribution. However, we stress that the algorithm will undo any split made in error. In addition, because this estimator does not include a finite population correction as such, when the overall sampling proportion $\frac{n}{N} = 1$ (which means that the algorithm is operating on the full data partition), we instead reject H_0 when $Var(R_i, S) \geq v$, according to the definition of variance diversity.

3.2 Discussion

Partitionings produced by Rothko-S are always guaranteed to satisfy the given anonymity requirement(s), provided that the entire input database satisfies the requirement(s). In virtually all cases (when the sample size is not extremely small) the resulting partitioning is also minimal (see Section 5). Potential non-minimality can, however, occur in the following scenario: Suppose the algorithm is operating on only a sample in some recursive instantiation (that is, R is larger than TM). If there does not exist a single (high-confidence) split that can be made during the growth phase, then it is possible that the resulting partitioning is non-minimal.[9] In this sense, the potential for non-minimality can be roughly equated with the power of the test. Similarly, if all splits made during the growth phase are undone during the pruning phase, we stop the algorithm to avoid thrashing.

There are two other important issues to consider. First, as we mentioned previously, our application can withstand some amount of imprecision and bias in the hypothesis test routine because splits that are made incorrectly based on a sample are eventually undone. However, it is important for efficiency that this does not happen too often. We continue to explore this issue in Section 5.6 as part of the experimental evaluation.

The second important issue to consider is the precision of the sampling-based algorithm with respect to workload-oriented splitting heuristics (InfoGain and Regression). It is clear that the split chosen using sample r is not guaranteed to be the same as the split that would be chosen according to the full partition R. This problem has been studied in the context of a sampling-based decision-tree construction algorithm (BOAT) [8], and could be similarly addressed in the anonymization setting using bootstrapping for splits, and subsequent refinement.[10]

[8] $\chi^2_\alpha\ (ndf)$ is the number such that the area beneath the chi-square density function (with n degrees of freedom) to the right is α.

[9] Of course, in the rare event that this scenario arises in practice, it is easily detected. For k-anonymity, ℓ-diversity, Median and InfoGain splitting, a reasonable implementation would simply switch to Rothko-T for the offending partition.

[10] The techniques proposed as part of BOAT would also have to be extended to handle the case where the entire partition tree does not fit in memory.

|R|	*Number of disk blocks in input relation R*
||R||	*Number of data tuples in input relation R*
TM	*Number of data tuples that fit in memory*
F_CACHE	*Number of frequency groups that fit in memory* (≥ 1)
height	*Height of the partition tree before each leaf partition fits in memory*

Figure 5: Notation for analytical comparison

From a practical perspective, however, we find that it is less important in our problem to choose the optimal split (according to the population) at every step. While decision trees typically seek to construct a compact structure that expresses an underlying concept, the anonymization algorithm continues partitioning the domain space until no allowable splits remain. We return to the issue of sampling and data quality in the experimental evaluation (Section 5.5).

4. ANALYTICAL COMPARISON

In order to lend insight to the experimental evaluation, this section provides a brief analytical comparison of the I/O behavior of Rothko-T and Rothko-S. For simplicity, we make this comparison for numeric data (binary splits), k-anonymity, and partition trees that are balanced and complete.[11] We use the notation described in Figure 5, and count the number of disk blocks that are read and written during the execution of each algorithm.

Rothko-T We begin with Rothko-T. Recall that once each leaf contains $\leq TM$ tuples, we switch to the in-memory algorithm. The height of the partition tree, prior to this switch, is easily computed. (We assume that $k \ll TM$.)

$$height = max\left(0, \left\lceil \log_2 \left(\frac{\| R \|}{TM} \right) \right\rceil \right)$$

Regardless of the available memory, the algorithm must scan the full data set $height + 1$ times. (The final scan imports the data in each leaf before executing the in-memory algorithm.) As F_CACHE increases, an increasing number of "repartitions" are eliminated.[12] Thus, the total number of reads and writes (disk blocks) is as follows:

$$\begin{aligned} repartitions_T &= \left\lceil \frac{height}{\lfloor \log_2(F\_CACHE) \rfloor + 1} \right\rceil \\ reads_T &= |R| * (height + repartitions_T + 1) \\ writes_T &= |R| * repartitions_T \end{aligned}$$

It is important to note that, unlike scalable decision trees [9], Rothko-T does not scale linearly with the size of the data. The reason for this is simple: decision trees typically express a "concept" of fixed size, independent of the size of the training data. In the

[11] Obviously, these assumptions do not hold in all cases. Under Median splitting, the partition tree will be only approximately balanced and complete due to duplicate values; for InfoGain and Regression splitting, the tree is not necessarily balanced and complete. Under ℓ-diversity or variance diversity, the analysis additionally depends on the distribution of sensitive attribute S. Nonetheless, the analytical comparison provides valuable intuition for the relative performance of the two scalable algorithms.

[12] For simplicity, we assume that the size of a frequency group is approximately constant for a given data set. In reality, the number of unique values per partition decreases as we descend in the tree.

anonymization algorithm, however, the height of the partition tree grows as a function of the input data and parameter k. For Median partitioning, the height of the full partition tree is approximately $\left\lfloor \log_2 \left(\frac{\|R\|}{k}\right)\right\rfloor$.

Rothko-S In the case of Rothko-S, the number of repartitions is instead a function of the estimator (rather than F_CACHE). The following recursive function counts the number of times the full data set is repartitioned under k-anonymity:

repartitions$_S(N)$
if $(N \leq TM)$
return 0
else
$p_0 = k/N$
$n = min(TM, N)$
$levels = max\left(x \in \mathbb{Z} : \frac{1}{2^x} - p_0 \geq z_{\alpha/2}\sqrt{\frac{p_0(1-p_0)}{n}\left(\frac{N-n}{N-1}\right)}\right)$
if $(levels > 0)$
return $1+$ repartitions$_S\left(\frac{N}{2^{levels}}\right)$
else // *non-minimal partitioning*
return 0

The data is scanned once to obtain the initial sample. Each time the data is repartitioned, the entire data set is scanned, and the new partitions written to disk. Then, each of the resulting partitions is scanned to obtain the random sample. Thus, the total number of reads and writes (disk blocks) is as follows:

$$\begin{aligned} reads_S &= |R| * (2 * repartitions_S(\| R \|) + 1) \\ writes_S &= |R| * repartitions_S(\| R \|) \end{aligned}$$

In practice, we observe that for reasonably large TM (large sample size), the total number of repartitions is often just 1. In this case, the entire data set is read three times, and written once.

5. EXPERIMENTAL EVALUATION

We conducted an analytical and experimental evaluation, intended to address the following high-level problems:

- **Need for Scalable Algorithm** We first seek to demonstrate the need to explicitly manage memory and I/O when anonymizing large data sets. (Section 5.2)
- **Evaluate and Compare Algorithms** One of our main goals is to evaluate and compare the our scalable algorithms (Rothko-T and Rothko-S). To this end, we perform an extensive experimental comparison of I/O behavior (Section 5.3) and total execution time (Section 5.4).
- **Sampling and Data Quality** When using a sample, the splits chosen according to the InfoGain and Regression split heuristics may not be identical to those chosen using the entire data set. Section 5.5 evaluates the practical implications.
- **Evaluate Hypothesis Tests** Our final set of experiments (Section 5.6) evaluates the effectiveness of the optimistic hypothesis-based splitting approach using a sample. By measuring the frequency of pruning, we show that the approach is quite effective. Also, though just rules of thumb, the tests described in Section 3.1 work quite well.

CentOS Linux (xfs file system)
512 MB memory
Intel Pentium 4 2.4 GHz processor
40 GB Maxtor IDE hard drive
(measured 54 MB/sec sequential bandwidth)
gcc version 3.4.4

Figure 6: Experimental system configuration

5.1 Experimental Setup

We implemented each of the scalable algorithms using C++. In all cases, disk-resident data partitions were stored as ordinary files of fixed-width binary-encoded tuples. File reads and writes were buffered into 256K blocks. Figure 6 describes our hardware/software configuration. In each of the experiments, we used a dedicated machine, with an initially cold buffer cache.

Our performance experiments used a synthetic data generator based on the generator of Agrawal et al. [4]. The quasi-identifier attributes were generated following the data distributions described in Figure 7, and when necessary, target attributes were generated as a function of these attributes. Functions C2 and C7 were used for classification tasks, while Functions R7 and R10 were used for regression tasks. Each quasi-identifier was treated as numeric (without user-defined generalization hierarchies), and each tuple was 44 bytes.

For the synthetic data, the size of an AV group (Median splitting) was approximately 8.1 MB. Because the class label attribute has two distinct values, the size of an AVC group (InfoGain splitting) was approximately 16.2 MB Also, for the sampling-based algorithm, we fixed $\alpha = 0.05$ throughout the experimental evaluation.

In addition to the synthetic data, our data quality experiments (Section 5.5) make use of the Census database, which contains several categorical attributes (and corresponding generalization hierarchies) [14].

5.2 Need for a Scalable Algorithm

When applied naively to large data sets, the Mondrian algorithms [13, 14] will often lead to thrashing, and the expected poor performance. To illustrate the need to explicitly manage memory and I/O, we performed a simple experiment. We ran our in-memory implementation (also in C++), allowing the virtual memory system to manage memory and I/O. Figures 8 and 9 show I/O behavior and runtime performance, respectively, for Median splitting and k-anonymity ($k = 1000$). As expected, the system begins to thrash for data sets that do not fit entirely in memory. These figures show performance for data sets containing up to 10 million records; in the remainder of this section, we will show that the scalable algorithms are easily applied to much larger data sets.

5.3 Counting I/O Requests

We begin the experimental comparison by focusing on the I/O incurred by each of the two proposed algorithms. Each of the experiments in this section uses Linux `/proc/diskstats` to count the total number of I/O requests (in 512 byte blocks) issued to the disk. We also compare the experimental measurements to the values predicted in Section 4.

All of the experiments in this section use Median partitioning and k-anonymity. The first two experiments each used $50\ million$ input tuples, and $k = 1000$. For Rothko-T, we fixed $TM = 2\ million$, and varied parameter F_CACHE. The results are shown in Figure 10. As expected, increasing F_CACHE reduces the number of I/O requests. However, the marginal improvement

Attribute	Distribution
salary	Uniform integer in [20,000, 150,000]
commission	If salary $\geq$ 75,000, then 0 Else Uniform integer in [10,000, 75,000]
age	Uniform integer in [20,80]
elevel	Uniform integer in [0, 4]
car	Uniform integer in [1, 20]
zipcode	Uniform integer in [1, 9]
hvalue	zipcode * h * 100,000 where h uniform in [0.5, 1.5]
hyears	Uniform integer in [1, 30]
loan	Uniform integer in [0, 500,000]

	Target Function T
C2	**if** $((age < 40) \wedge (50K \leq salary \leq 100K)) \vee$ $((40 \leq age < 60) \wedge (75K \leq salary \leq 125K)) \vee$ $((age \geq 60) \wedge (25K \leq salary \leq 75K))$ **then** $T = A$ **else** $T = B$
C7	$disposable = 0.67 \times (salary + commission)$ $-0.2 \times loan - 20K$ **if** $disposable > 0$ **then** $T = A$ **else** $T = B$
R7	$T = 0.67 \times (salary + commission)$ $-0.2 \times loan - 20K$
R10	**if** $hyears < 20$ **then** $equity = 0$ **else** $equity = 0.1 \times hvalue \times (hyears - 20)$ $T = 0.67 \times (salary + commission)$ $-5000 \times elevel + 0.2 \times equity - 10K$

Figure 7: Synthetic data generator

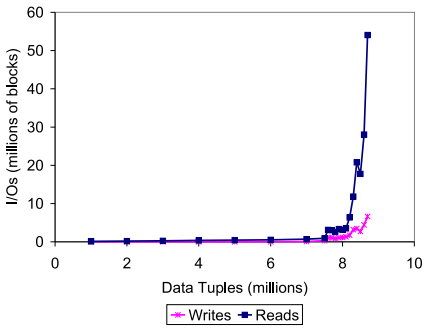


Figure 8: In-Memory Mondrian I/O

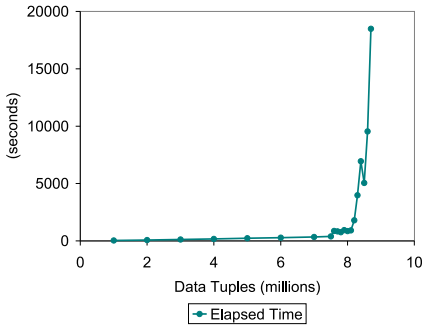


Figure 9: In-Memory Mondrian time

obtained from each additional frequency group is decreasing. In some cases, the observed number of disk reads is smaller than expected due to file system buffering.

For Rothko-S, we varied the sample size, and the results are shown in Figure 11. Notice that for this wide range of sample sizes, the data was repartitioned just once, meaning that the algorithm read the entire data set 3 times, and wrote it once. Also, the total amount of I/O is substantially less than that of Rothko-T.

Finally, we performed a scale-up experiment, increasing the data size, and fixing $TM = 2\ million$, $k = 1000$. The total number of I/O requests (reads and writes) are shown in Figure 12. Again, Rothko-T is able to exploit the buffer cache to some extent, but the total amount of I/O is substantially more than Rothko-S.

5.4 Runtime Performance

Perhaps most importantly, we evaluated the runtime performance of both proposed algorithms. All of the experiments in this section use k-anonymity as the anonymity requirement. In each case, we break down the execution time into three components: (1) User space CPU time, (2) Kernel space CPU time, and (3) I/O wait time. These statistics were gathered from the system via `/proc/stat`. Note that in all cases, the experiments in this section produced minimal partitionings.

We begin with Median partitioning. The first set of experiments measured scale-up performance, fixing $TM = 2\ million$, and $k = 1000$. Figures 13 and 14 show the results for Rothko-T, with $F\_CACHE = 1$ and 8, respectively. Figure 15 shows the results for Rothko-S. As expected, the sampling-based algorithm was faster, both in terms of total execution time and CPU time. Additionally, each of the algorithms goes through periods where execution is I/O-bound. Interestingly, the I/O wait times are similar for Rothko-T ($F\_CACHE = 8$) and Rothko-S. However, this is deceptive. Although Rothko-T does more I/O, it also performs more in-memory calculations, thus occupying the CPU while the file system flushes the buffer cache asynchronously.

The second set of experiments considered the effects of parameter k. Results for these experiments are shown in Figures 16, 17, and 18. As expected, a decreasing value of k leads to more computation. However, because the algorithms all switch to the in-memory algorithm after some number of splits, this additional cost falls to the CPU.

Finally, we compared scale-up performance using the InfoGain split criterion, again fixing $TM = 2\ million$, and $k = 1000$. For these experiments, we used label function C2 to generate the class labels. Figures 19 and 20 show the results for Rothko-T ($F\_CACHE = 1, 4$). Figure 15 shows the results for Rothko-S. As expected, the CPU cost incurred by these algorithms is greater than Median partitioning, particularly due to the extra cost of finding allowable numeric thresholds that maximize information gain. However, Rothko-S consistently outperforms Rothko-T.[13]

5.5 Effects of Sampling on Data Quality

In Section 3.2, we discussed some of the potential shortcomings of the sampling-based algorithm, and we noted that one primary concern is imprecision with respect to the InfoGain and Regression split criteria. In this section, we evaluate the effects of sampling with respect to data quality. For reasonably large sample sizes, we find that in practice the effect is often minimal.

[13]For efficiency, the recursive partitioning procedure switched to Median partitioning when the information gain resulting from a new split dipped below a 0.01. We note that continuing the InfoGain splitting all the way to the leaves is very CPU-intensive, particularly for numeric attributes, because of the required sorting.

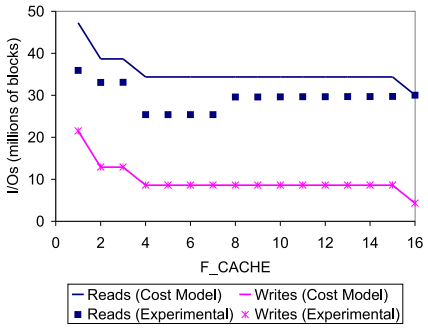


Figure 10: Rothko-T I/O

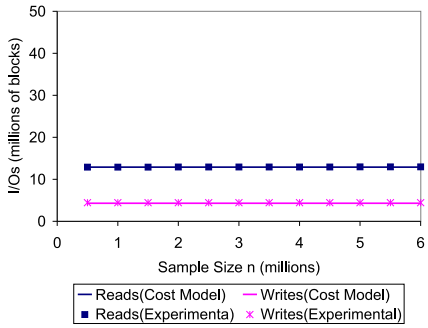


Figure 11: Rothko-S I/O

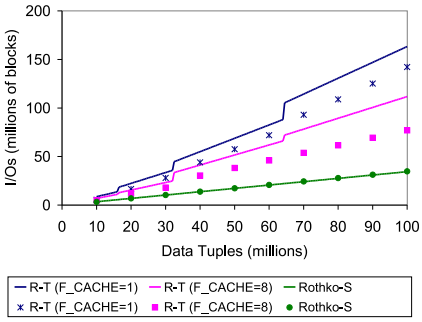


Figure 12: Scale-up I/O

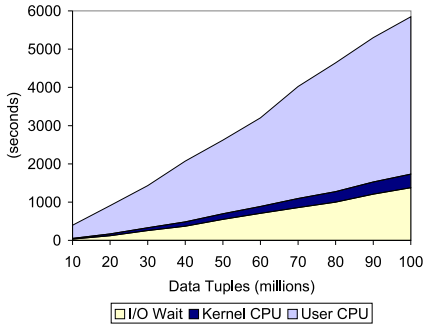


Figure 13: Rothko-T (F_CACHE = 1) Median Splitting

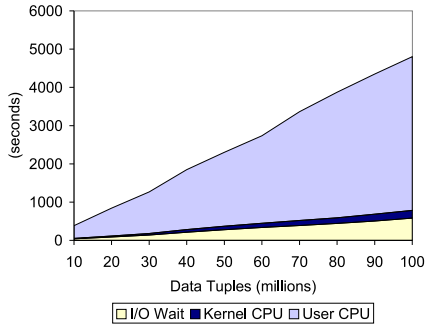


Figure 14: Rothko-T (F_CACHE = 8) Median Splitting

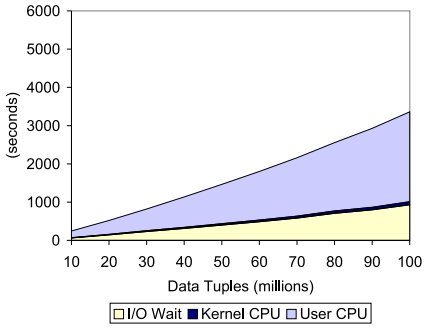


Figure 15: Rothko-S Median Splitting

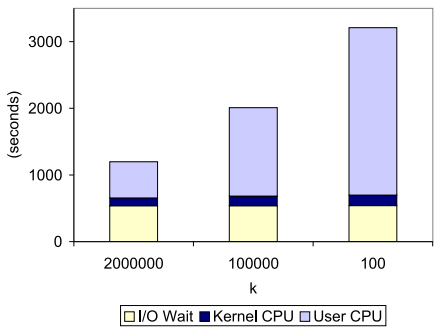


Figure 16: Rothko-T (F_CACHE = 1) Median Splitting

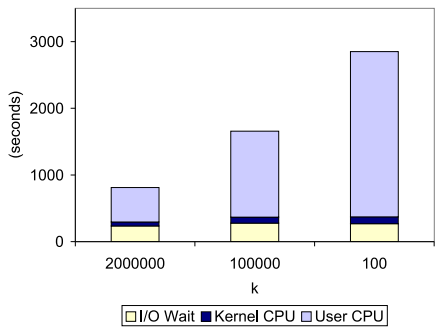


Figure 17: Rothko-T (F_CACHE = 8) Median Splitting

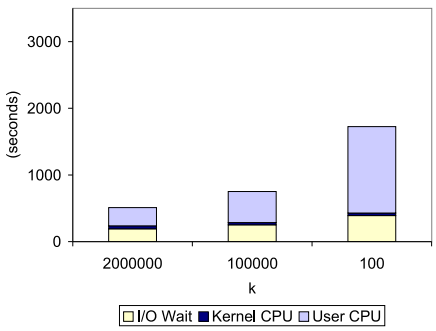


Figure 18: Rothko-S Median Splitting

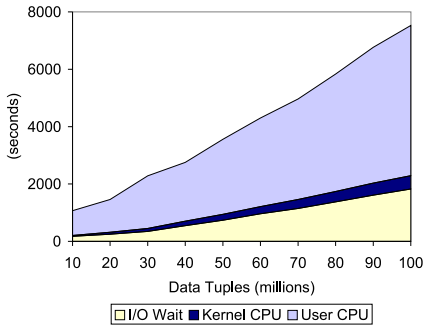


Figure 19: Rothko-T (F_CACHE = 1) InfoGain Splitting

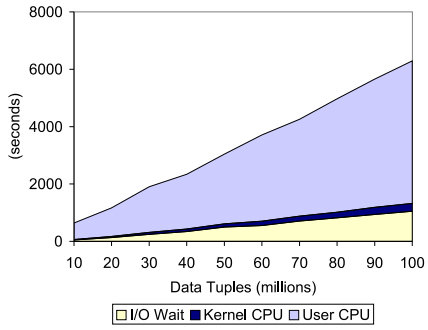


Figure 20: Rothko-T (F_CACHE = 4) InfoGain Splitting

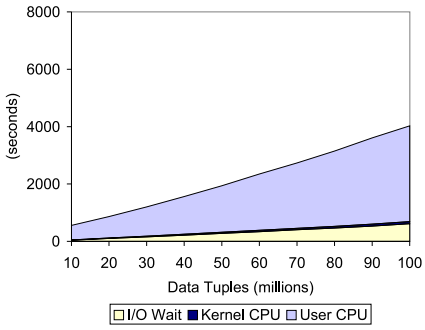


Figure 21: Rothko-S InfoGain Splitting

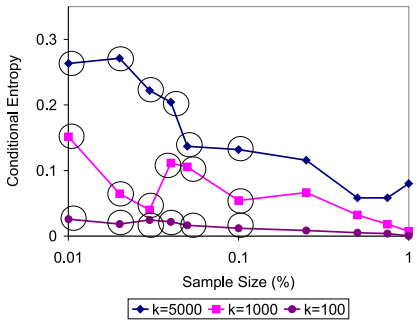


Figure 22: C2

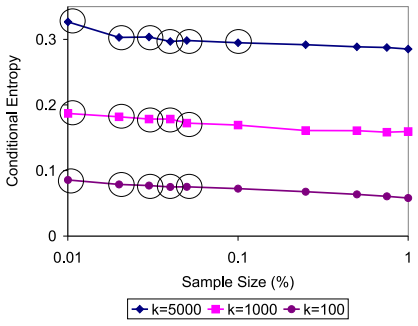


Figure 23: C7

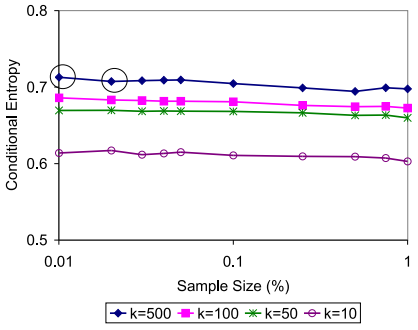


Figure 24: Census Classification

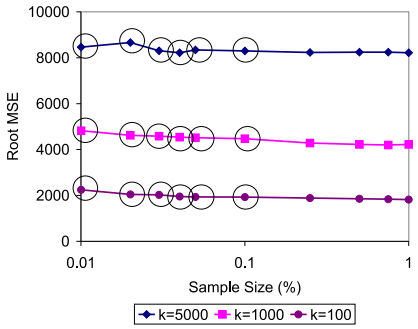


Figure 25: R7

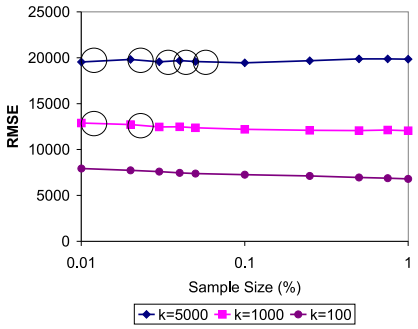


Figure 26: R10

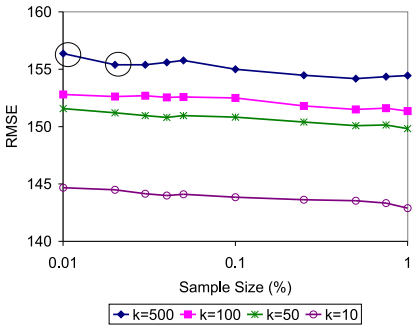


Figure 27: Census Regression

There are a number of different ways to measure data quality. In the interest of simplicity, in these experiments, when using InfoGain splitting, we measured the quality of the resulting anonymized data using the conditional entropy of the class label (C), with respect to the partitioning [14]. For Regression splitting, we measure the root mean squared error (RMSE) that would result if a data recipient used the mean value in each equivalence class to estimate the values of a numeric target attribute T. Both of these measures relate directly to the respective greedy split criteria.

$$RMSE(T) = \sqrt{\frac{1}{|R|}\sum_{i=1}^{m} |R_i| * Var(R_i, T)}$$

We performed experiments using both synthetic and real-life data. Results for synthetic data and InfoGain splitting are shown in Figures 22 and 23. Results for Regression splitting are shown in Figures 25 and 26. For each experiment, we generated 10 data sets (each containing 100,000 records), and we increased the sample size. The reported results are averaged across the ten data sets. In the figure, we circled partitionings that are potentially non-minimal.

Increasing the sample size does lead to small improvement in quality (decreased entropy or error). However, for reasonably large sample sizes, the difference is very small. In all of our experiments the sample size had a much smaller impact on quality than the anonymity parameter k.

We also conducted a similar experiment using the Census database [14]. Results are shown in Figures 24 and 27, for InfoGain and Regression splitting, respectively, again denoting potentially non-minimal partitionings with an additional circle. Again, the improvement in quality gained from increasing the sample size is small.

5.6 Hypothesis Tests and Pruning

One of the important components in the design of the sampling-based algorithm is choosing an appropriate means of checking each anonymity requirement(k-anonymity, ℓ-diversity, and variance diversity) using a sample. Although the algorithm will always undo splits made in error, it is important to have a reasonable procedure in order to avoid excessive pruning.

In this section, we evaluate the effectiveness of the hypothesis-based approach described in Section 3.1. As mentioned previously, our hypothesis tests for ℓ-diversity and variance diversity are just "rules of thumb". Nonetheless, we find that the approach of using a hypothesis test, as well as the specific tests outlined in Section 3.1, actually work quite well in practice.

We again used the synthetic data generator (Figure 7), and the Median split criterion. For each experiment, we used an input of 100,000 tuples, and varied the sample size. For each experiment, we repartitioned the data automatically when the height of the tree reached 8 (due to memory limitations for storing sensitive value histograms under variance diversity).

We conducted experiments using k-anonymity, ℓ-diversity, and variance diversity, each as the sole anonymity requirement in the respective experiment. For ℓ-diversity, we used *zipcode* as the sensitive attribute, and fixed $c = 1$. For variance diversity, we used *salary* as the sensitive attribute. In addition to the uniform salary distribution, we also considered a normal distribution. (The population variance of the uniform *salary* is approximately 1.4e9; the population variance of the normal *salary* is approximately 1.1e8.)

Figure 28 shows our results. Each entry indicates the total number of nodes that were pruned during the algorithm's entire execution. The numbers in parentheses indicate the number of nodes that are pruned when we use a naive approach that does not incorporate hypothesis tests (see Section 3.1). An "x" indicates that the resulting partitioning was (potentially) non-minimal, as described

n \ *k*	10	100	1000	10000
100	68 (1384)	x (x)	x (x)	x (x)
250	30 (1110)	7 (97)	x (x)	x (x)
500	11 (419)	12 (55)	x (x)	x (x)
1000	0 (0)	5 (6)	x (x)	x (x)
2500	0 (0)	2 (1)	1 (3)	x (x)
5000	0 (0)	0 (0)	1 (4)	x (x)
10000	0 (0)	0 (0)	0 (0)	x (x)
25000	0 (0)	0 (0)	0 (0)	0 (0)

(a) k-Anonymity

n \ ℓ	2	4	6	8
100	97 (631)	x (x)	x (x)	x (x)
250	65 (87)	x (x)	8 (67)	x (x)
500	2 (763)	x (111)	2 (32)	x (x)
1000	112 (91)	1 (170)	1 (33)	x (16)
2500	0 (0)	0 (2)	1 (0)	0 (2)
5000	0 (0)	0 (2)	0 (0)	0 (9)
10000	0 (0)	0 (1)	0 (0)	0 (0)
25000	0 (0)	0 (1)	0 (0)	0 (0)

(b) ℓ-Diversity

n \ v	1.1e9	1.2e9	1.3e9
100	x (x)	x (x)	x (x)
250	x (510)	x (x)	x (x)
500	x (443)	x (434)	x (x)
1000	0 (456)	x (372)	x (x)
2500	0 (0)	0 (87)	x (146)
5000	0 (0)	0 (0)	0 (47)
10000	0 (0)	0 (0)	0 (5)
25000	0 (0)	0 (2)	0 (7)

(c) Variance Diversity (Uniform S)

n \ v	7e7	8e7	9e7
100	x (x)	x (x)	x (x)
250	x (396)	x (x)	x (x)
500	x (599)	x (457)	x (x)
1000	0 (402)	0 (405)	x (278)
2500	0 18)	0 (66)	x (169)
5000	0 (0)	0 (0)	0 (6)
10000	0 (0)	0 (0)	0 (13)
25000	0 (0)	0 (0)	0 (17)

(d) Variance Diversity (Normal S)

Figure 28: Pruning and non-minimality in Rothko-S

in Section 3.2.

There are two important things to note from these results. First and foremost, the estimates are reasonably well-behaved, and do not lead to an excessive amount of pruning, even for small samples. Similarly, although our hypothesis tests are just rules of thumb, they provide for much cleaner execution (less pruning) than the naive approach of using no hypothesis test.

As expected, the incidence of both non-minimality and pruning decreases with increased sample size.

6. RELATED WORK

In recent years, numerous algorithms have been proposed for k-anonymous generalization [2, 5, 7, 10, 12, 16, 19, 20], and the related problems of anonymous clustering and microaggregation [1, 3, 6, 24], but few have considered data sets larger than main memory.

The Incognito algorithm operated on external (disk-resident) data [12]. However, the complexity of the algorithm was exponential in the size of the attribute schema, making it impractical in many situations. Additionally, an empirical study indicates that the full-domain recoding technique may produce lower-quality data in practice [13].

In order to provide k-anonymity in the context of location-based services, Mokbel et al. proposed using a scalable grid-based structure [17]. They describe two algorithms: a batch algorithm (which operates bottom-up), and an algorithm for incremental updates. Neither of the algorithms was designed to incorporate additional anonymity requirements (e.g., ℓ-diversity) or workload-oriented splitting heuristics (e.g., InfoGain splitting). The proposed techniques were also designed to handle 2-dimensional spatial data, and it is not immediately clear how they would scale to data with higher dimensionality.

To the best of our knowledge, all of the other proposed algorithms were designed to handle only memory-resident data, and none has been evaluated with respect to data substantially larger than the available memory. Nonetheless, many interesting algorithmic approaches have been considered, including optimal search in cube space [12, 19], optimal search in a more flexible space of generalizations [5], genetic algorithms [10], approximation algorithms [2, 16], and greedy heuristic search [7, 13, 14, 20, 22, 24].

There have also been a number of recent extensions and variations of the k-anonymity model, including ℓ-diversity [15], described in Section 1.1. Additionally, Xiao and Tao proposed generalizing sensitive values, and allowing individuals to determine at what level of granularity these attributes are exposed [23]. Finally, Kifer and Gehrke proposed releasing multiple (generalized) marginal tables, the schema of which forms a decomposable graph, as a technique for combating high dimensionality [11]. Future work might consider techniques for integrating these and other extensions into the scalable framework described in this paper.

7. CONCLUSION & FUTURE WORK

This paper considered scaling an existing generalization-based anonymization framework (Mondrian [13, 14]) to data sets that are much larger than main memory, and proposed two separate techniques. The first, Rothko-T, is based on ideas from the RainForest framework for scalable decision tree construction [9]. Given modern memory sizes, Rothko-T can often be applied when the anonymity requirement is k-anonymity and/or ℓ-diversity, and the split criterion is either Median or InfoGain. The output of this technique is guaranteed to satisfy all given anonymity requirements, and it is also guaranteed to be a minimal partitioning.

The second technique, called Rothko-S, is also guaranteed to produce output that satisfies all given anonymity constraints (provided that the full input data set satisfies these requirements). This recursive algorithm uses a data sample, and partitions the space "optimistically" using the sample and a set of estimators. Periodically, the algorithm repartitions the data, and prunes away any splits that were made in error.

We conducted an experimental performance evaluation, and found

that the sampling-based algorithm is often more efficient, both in terms of I/O and elapsed time. In addition, we found that for reasonably-large sample sizes, choosing workload-oriented splits (InfoGain or Regression) based on a sample has only a small impact on the quality of the output data.

Finally, there are several interesting opportunities for future work. Specifically, in this paper, we considered using sampling as a way to scale an anonymization algorithm to data sets larger than main memory. The hypothesis tests we developed (Section 3.1) for k-anonymity, ℓ-diversity, and variance diversity are reasonable rules of thumb, particularly for the large samples encountered in the external algorithm. However, if we had a more precise set of tests (and precise characterizations of power and significance levels), it is reasonable to believe that we could also apply a sampling-based algorithm to enhance the performance of the in-memory case, choosing sample sizes in accordance with the given test(s).

8. REFERENCES

[1] C. Aggarwal and P. Yu. A condensation approach to privacy-preserving data mining. In *Proceedings of the 9th International Conference on Extending Database Technology*, 2004.

[2] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Anonymizing tables. In *Proceedings of the 10th International Conference on Database Theory*, January 2005.

[3] G. Aggarwal, T. Feder, K. Kenthapadi, R. Panigrahy, D. Thomas, and A. Zhu. Achieving anonymity via clustering in a metric space. In *Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2006.

[4] R. Agrawal, S. Ghosh, T. Imielinski, and A. Swami. Database mining: A performance perspective. In *IEEE Transactions on Knowledge and Data Engineering*, volume 5, 1993.

[5] R. Bayardo and R. Agrawal. Data privacy through optimal k-anonymity. In *Proceedings of the 21st International Conference on Data Engineering*, April 2005.

[6] J. Domingo-Ferrer and J. Mateo-Sanz. Practical data-oriented microaggregation for statistical disclosure control. *IEEE Transactions on Knowledge and Data Engineering*, 4(1), 2002.

[7] B. Fung, K. Wang, and P. Yu. Top-down specialization for information and privacy preservation. In *Proceedings of the 21st International Conference on Data Engineering*, April 2005.

[8] J. Gehrke, V. Ganti, R. Ramakrishnan, and W. Loh. BOAT: Optimistic decision tree construction. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1999.

[9] J. Gehrke, R. Ramakrishnan, and V. Ganti. RainForest: A framework for fast decision tree construction of large datasets. In *Proceedings of the 24th International Conference on Very Large Databases*, 1998.

[10] V. Iyengar. Transforming data to satisfy privacy constraints. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2002.

[11] D. Kifer and J. Gehrke. Injecting utility into anonymized datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2006.

[12] K. LeFevre, D.DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2005.

[13] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering*, 2006.

[14] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Workload-aware anonymization. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[15] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-Diversity: Privacy beyond k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering*, 2006.

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

[17] M. Mokbel, C. Chow, and W. Aref. The new casper: Query processing for location services without compromising privacy. In *Proceedings of the 32nd International Conference on Very Large Databases*, 2006.

[18] J. A. Rice. *Mathematical Statistics and Data Analysis*. Wadsworth, Inc., 1995.

[19] P. Samarati. Protecting respondants' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6), November/December 2001.

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

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

[22] K. Wang, P. Yu, and S. Chakraborty. Bottom-up generalization: A data mining solution to privacy protection. In *Proceedings of the 4th IEEE International Conference on Data Mining*, November 2004.

[23] X. Xiao and Y. Tao. Personalized privacy preservation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2006.

[24] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A. Fu. Utility-based anonymization using local recoding. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

APPENDIX

Given an input tuple set R, consider the set of all possible *bucketizations* (groupings of tuples into equivalence classes). It is possible to define a partial order $\preceq$ on this set of bucketizations, where $R^* \preceq R^{**}$ if and only if every bucket in R^{**} is the union of one or more of the buckets in R^*.

Let R^* and R^{**} be bucketizations of input data set R such that $R^* \preceq R^{**}$, and let $allowable_{def}(R^*)$ denote that bucketization R^* is allowable with respect to privacy requirement def. Privacy requirement def is *monotone* if and only if $allowable_{def}(R^*) \rightarrow allowable_{def}(R^{**})$.

THEOREM 1. *Variance diversity is monotone.*

PROOF. Without loss of generality, consider two finite multisets of real numbers, $A = \{a_1, ..., a_m\}$ and $B = \{b_1, ..., b_n\}$, which are the sensitive values in two non-overlapping buckets (equivalence classes). Suppose that each bucket satisfies variance diversity. That is, $Var(A) \geq v$ and $Var(B) \geq v$. To show that $Var(A \cup B) \geq v$, it is sufficient to show that $Var(A \cup B) \geq min(Var(A), Var(B))$.

For two random variables, X and Y, the Law of Total Variance [18] states that $Var(X) = E[Var(X|Y)] + Var[E(X|Y)]$. For finite sets A and B, we use Y to indicate membership in one of the two sets. Thus, we can compute $Var(A \cup B)$ as follows, where $\overline{A}$ denotes the mean value in set A:

$$\begin{aligned} \overline{R} &= \frac{m}{m+n}\overline{A} + \frac{n}{m+n}\overline{B} \\ Var(A \cup B) &= \frac{m}{m+n}Var(A) + \frac{n}{m+n}Var(B) \\ &\quad + \frac{m}{m+n}(\overline{A} - \overline{R})^2 + \frac{n}{m+n}(\overline{B} - \overline{R})^2 \end{aligned}$$

Thus, $Var(A \cup B) \geq min(Var(A), Var(B))$. $\square$