Efficient Algorithms for Masking and Finding Quasi-Identifiers

Rajeev Motwani
Stanford University
rajeev@cs.stanford.edu

Ying Xu
Stanford University
xuying@cs.stanford.edu

ABSTRACT

A quasi-identifier refers to a subset of attributes that can uniquely identify most tuples in a table. Incautious publication of quasi-identifiers will lead to privacy leakage. In this paper we consider the problems of finding and masking quasi-identifiers. Both problems are provably hard with severe time and space requirements. We focus on designing efficient approximation algorithms for large data sets.

We first propose two natural measures for quantifying quasi-identifiers: distinct ratio and separation ratio. We develop efficient algorithms that find small quasi-identifiers with provable size and separation/distinct ratio guarantees, with space and time requirements sublinear in the number of tuples. We also design practical algorithms for finding all minimal quasi-identifiers. Finally we propose efficient algorithms for masking quasi-identifiers, where we use a random sampling technique to greatly reduce the space and time requirements, without much sacrifice in the quality of the results. Our algorithms for masking and finding minimum quasi-identifiers naturally apply to stream databases. Extensive experimental results on real world data sets confirm efficiency and accuracy of our algorithms.

1. INTRODUCTION

In the age of globalization, governments, companies and organizations are publishing and sharing more and more micro-data for research or business purpose. This puts personal privacy at risk. As pointed out in the seminal paper by Sweeney [27], naively removing identifying attributes such as name and SSN leaves open attacks that combine the data with other publicly available information to identify represented individuals. A well-known example is that the combination of gender, date of birth, and zipcode can uniquely determine about 87% of the population in United States. Such an identity-leaking attribute combination is often referred to as a *quasi-identifier*. In other words, a quasi-identifier is a subset of attributes that can uniquely identify most tuples in a table.

To avoid such linking attacks via quasi-identifiers, the concept of k-anonymity was proposed by Sweeney [27, 26] and many algorithms for k-anonymity have been developed [25, 2, 5]. In this paper we consider the problem of masking quasi-identifiers: we want to publish a subset of attributes (we either publish the exact value of every tuple on an attribute, or not publish the attribute at all), so that no quasi-identifier is revealed in the published data. This can be viewed as a variant of k-anonymity where the suppression is only allowed at the attribute level. There are two reasons we consider this problem. First, the traditional tuple-level suppression may distort the distribution of the original data and the association between attributes, so sometimes it is more desirable to publish fewer attributes with complete and accurate information. Second, as noted in [16], the traditional k-anonymity algorithms are expensive and do not scale well to large data sets; by restricting the suppression to a coarser level, we are able to design much more efficient algorithms.

We also consider the problem of finding minimum or minimal keys and quasi-identifiers, which can be used by adversaries to perform linking attacks. When a table which is not properly anonymized is published, an adversary would be interested in finding keys or quasi-identifiers in the table so that once he collects other persons' information on those attributes, he will be able to link the records to real world entities. Collecting information on each attribute incurs certain cost to the adversary (for example, he needs to look up yellow pages to collect the area code of phone numbers, to get party affiliation information from the voter list, etc), so the adversary wishes to find a small subset of attributes that is a key or almost a key to minimize the attack cost.

Finding keys and quasi-identifiers also has other important applications besides privacy. One application is data cleaning. Integration of heterogeneous databases sometimes causes the same real-world entity to be represented by multiple records in the integrated database due to spelling mistakes, inconsistent conventions, etc. A critical task in data cleaning is to identify and remove such fuzzy duplicates [4, 7]. We can estimate the ratio of fuzzy duplicates, for example by checking some samples manually or plotting the distribution of pairwise similarity; now if we can find a quasi-identifier whose "quasiness" is similar to the fuzzy duplicate ratio, then those tuples which collide on the quasi-identifier are likely to be fuzzy duplicates. Finally, quasi-identifiers are a special case of approximate functional dependency [14, 24, 12], and their automatic discovery is valuable to query optimization and indexing.

In this paper, we study the problems of finding and masking quasi-identifiers in given tables. Both problems are provably hard with severe time and space requirements, so we focus on designing efficient approximation algorithms for large data sets.

Before presenting the algorithms, we need to first define measures for quantifying the "quasiness" of quasi-identifiers. We propose two natural measures – separation ratio and distinct ratio.

Consider the problem of finding the minimum key. The problem is NP-hard and the best-known approximation algorithm is a greedy algorithm with approximation ratio $O(\ln n)$ (n is the number of tuples); however, even this greedy algorithm requires multiple scans of the table, which is expensive for large databases that cannot reside in main memory and prohibitive for stream databases. To enable more efficient algorithms, we sacrifice accuracy by allowing approximate answers (quasi-identifiers). We develop efficient algorithms that find small quasi-identifiers with provable size and separation/distinct ratio guarantees, with both space and time complexities sublinear in the number of input tuples. We also design practical algorithms for identifying all minimal keys and quasi-identifiers.

Finally we present efficient algorithms for masking quasi-identifiers. We use a random sampling technique to greatly reduce the space and time requirements, without sacrificing much in the quality of the results.

Our algorithms for masking and finding minimum quasi-identifiers can be viewed as streaming algorithms: we only require one pass over the table and the space complexity is sublinear in the number of input tuples (at the cost of only providing approximate solutions), therefore these algorithms naturally apply to stream databases.

1.1 Definitions and Overview of Results

A *key* is a subset of attributes that uniquely identifies each tuple in a table. A *quasi-identifier* is a subset of attributes that can distinguish almost all tuples. We propose two natural measures for quantifying a quasi-identifier. Since keys are a special case of functional dependencies, our measures for quasi-identifiers also conform with the measures of approximate functional dependencies proposed in earlier work [14, 24, 12].

(1) An *α-distinct quasi-identifier* is a subset of attributes which becomes a key in the table remaining after the removal of at most a $1 - \alpha$ fraction of tuples in the original table.

(2) We say that a subset of attributes *separates* a pair of tuples x and y if x and y have different values on at least one attribute in the subset. An *α-separation quasi-identifier* is a subset of attributes which separates at least a α fraction of all possible tuple pairs.

We illustrate the notions with an example (Table 1). The example table has 3 attributes. The attribute *age* is a 0.6-distinct quasi-identifier because it has 3 distinct values in a total of 5 tuples; it is a 0.8-separation quasi-identifier because there are 10 distinct pairs of tuples and 8 pairs can be separated by *age*. Readers can verify that the attribute subset $\{sex, state\}$ is 0.8-distinct and 0.9-separation.

The distinct ratio and separation ratio of a quasi-identifier can be very different. The separation ratio of a quasi-identifier is usually much larger than its distinct ratio, but there is no one-to-one mapping. Let us consider a 0.5-distinct quasi-identifier in a table of 100 tuples. One possible scenario is that projected on the quasi-identifier, there are 50 distinct values and each value corresponds to 2 tuples. This quasi-identifier separates all but 50 pairs of tuples, hence is $1 - \frac{50}{\binom{100}{2}} \approx 0.99$-separation. The other possible scenario is that for 49 of the 50 distinct values, there is only one tuple for each value, and all the other 51 tuples have the same value. Then this 0.5-distinct quasi-identifier is 0.75-separation. Indeed, an α-distinct quasi-identifier can be an α'-separation quasi-identifier where α' can be as small as $2\alpha - \alpha^2$, or as large as $1 - \frac{2(1-\alpha)}{n}$. Both distinct ratio and separation ratio are very natural measures for quasi-identifiers and have different applications as noted in the literature on approximate functional dependency. In this paper we study quasi-identifiers using both measures.

	age	sex	state
1	20	Female	CA
2	30	Female	CA
3	40	Female	TX
4	20	Male	NY
5	40	Male	CA

Table 1: An example table. The first column labels the tuples for future references and is not part of the table.

Given a table with n tuples and m attributes, we consider the following problems. The *size* of a key (quasi-identifier) refers to the number of attributes in the key.

Minimum Key Problem: find a key of the minimum size. This problem is provably hard so we also consider its relaxed version:

(ϵ, δ)-Separation or -Distinct Minimum Key Problem: look for a quasi-identifier with a small size such that, with probability at least $1 - \delta$, the output quasi-identifier has separation or distinct ratio at least $1 - \epsilon$.

All Minimal Keys (or β-Separation Quasi-identifiers, or β-Distinct Quasi-identifiers) Problem: find all minimal keys (or β-separation quasi-identifiers, or β-distinct quasi-identifiers). A key is minimal if it does contain any other key.

β-Separation or -Distinct Quasi-identifier Masking Problem: delete a minimum number of attributes such that there is no quasi-identifier with separation or distinct ratio greater than β in the remaining attributes.

In the example of Table 1, $\{age, state\}$ is a minimum key, with size 2; all minimal keys are $\{age, state\}$, $\{age, sex\}$; the optimal solution to 0.8-distinct quasi-identifier masking problem is $\{sex, state\}$; the optimal solution to 0.8-separation quasi-identifier masking problem is $\{age\}$, $\{sex\}$ or $\{state\}$, all of size 1.

The result data after quasi-identifier masking can be viewed as an approximation to k-anonymity. For example, after 0.2-distinct quasi-identifier masking, the result data is approximately 5-anonymous, in the sense that on average each tuple is indistinguishable from another 4 tuples. It does not provide perfect privacy because there may still exist some tuple with a unique value, but nevertheless it provides anonymity for the majority of the tuples. The k-anonymity problem is NP-hard [19, 2]; further, Lodha and Thomas [16] note that there is no efficient approximation algorithm known that scale well for large tables, and they also use a notion similar to quasi-identifiers to design more efficient algorithms while preserve privacy for majority tuples. As in the traditional k-anonymity literature, we would like to maximize the utility of published data, and we measure utility in terms of the number of attributes published.

We summarize below the contributions of this paper.

1. We propose greedy algorithms for the (ϵ, δ)-separation and distinct minimum key problems, which find small quasi-identifiers with provable size and separation (distinct) ratio guarantees, with space and time requirements sublinear in n. In particular, the space complexity is $O(m^2)$ for the (ϵ, δ)-separation minimum key problem, and $O(m\sqrt{mn})$ for (ϵ, δ)-distinct. The algorithms are particularly useful when $n \gg m$, which is typical of database applications where a large table may consist of millions of tuples, but only a relatively small number of attributes. We also extend the algorithms to find the approximate minimum β-separation quasi-identifiers. (Section 2)
2. We design practical algorithms for finding all minimal keys, β-distinct, and β-separation quasi-identifiers. Our algorithms use small sample tables to efficiently prune the key space; experiments show that the pruning is able to improve the running time by orders of magnitude. (Section 3)
3. We present greedy algorithms for β-separation and β-distinct quasi-identifier masking. The algorithms are slow on large data sets, and we use a random sampling technique to greatly reduce the space and time requirements, without much sacrifice in the utility of the published data. (Section 4)
4. We have implemented all the above algorithms and conducted extensive experiments using real data sets. The experimental results confirm the efficiency and accuracy of our algorithms. (Section 5)

2. FINDING MINIMUM KEYS

In this section we consider the Minimum Key problem. First we show the problem is NP-hard (Section 2.1) and the best approximation algorithm is a greedy algorithm which gives $O(\ln n)$-approximate solution (Section 2.2). The greedy algorithm requires multiple scans of the table, which is expensive for large tables and inhibitive for stream databases. To enable more efficient algorithms, we relax the problem by allowing quasi-identifiers, i.e. the (ϵ, δ)-Separation (Distinct) Minimum Key problem. We develop random sampling based algorithms with approximation guarantees and sublinear space (Section 2.3, 2.4).

2.1 Hardness Result

The Minimum Key problem is NP-Hard, which follows easily from the NP-hardness of the *Minimum Test Collection* problem.

> *Minimum Test Collection:* Given a set S of elements and a collection C of subsets of S, a test collection is a subcollection of C such that for each pair of distinct elements there is some set that contains exactly one of the two elements. The Minimum Test Collection problem is to find a test collection with the smallest cardinality.

Minimum Test Collection is equivalent to a special case of the Minimum Key problem where each attribute is boolean: let S be the set of tuples and C be all the attributes; each subset in C corresponds to an attribute and contains all the tuples whose values are *true* in this attribute, then a test collection is equivalent to a key in the table. Minimum Test Collection is known to be NP-hard [9], therefore the Minimum Key problem is also NP-hard.

2.2 A Greedy Approximation Algorithm

The best known approximation algorithm for Minimum Test Collection is a greedy algorithm with approximation ratio $1+2\ln|S|$ [20], i.e. it finds a test collection with size at most $1+2\ln|S|$ times the smallest test collection size. The algorithm can be extended to the more general Minimum Key problem, where each attribute can be from an arbitrary domain, not just boolean.

Before presenting the algorithm, let us consider a naive greedy algorithm: compute the separation (or distinct) ratio of each attribute in advance; each time pick the attribute with the highest separation ratio in the remaining attributes, until get a key. The algorithm is fast and easy to implement, but unfortunately it does not perform well when the attributes are correlated. For example if there are many attributes pairwise highly correlated and each has a high separation ratio, then the optimal solution probably includes only one of them while the above greedy algorithm are likely picks all of them. The approximation ratio of this algorithm can be arbitrarily bad.

A fix to the naive algorithm is to pick each time the attribute which separates the largest number of tuple pairs not yet separated. To prove the approximation ratio of the algorithm, we reduce Minimum Key to the Minimum Set Cover problem. The reduction plays an important role in designing algorithms for finding and masking quasi-identifiers in later sections.

> *Minimum Set Cover:* Given a finite set S (called the *ground set*) and a collection C of subsets of S, a *set cover* I is a subcollection of C such that every element in S belongs to at least one member of I. *Minimum Set Cover* problem asks for a set cover with the smallest size.

Given an instance of Minimum Key with n tuples and m attributes, we reduce it to a set cover instance as follows: the ground set S consists of all distinct unordered pairs of tuples ($|S| = \binom{n}{2}$); each attribute c in the table is mapped to a subset containing all pairs of tuples separated by attribute c. Now a collection of subsets covers S if and only if the corresponding attributes can separate all pairs of tuples, i.e., those attributes form a key, therefore there is a one-to-one map between minimum set covers and minimum keys.

Consider the example of Table 1. The ground set of the corresponding set cover instance contains 10 elements where each element is a pair of tuples. The column *age* is mapped to a subset c_{age} with 8 pairs: $\{(1,2),(1,3),(1,5),(2,3),(2,4),(2,5),(3,4),(4,5)\}$; the column *sex* is mapped to a subset c_{sex} with 6 pairs, and *state* 7 pairs. The attribute set

$\{age, sex\}$ is a key; correspondingly the collection $\{c_{age}, c_{sex}\}$ is a set cover.

The *Greedy Set Cover Algorithm* starts with an empty collection (of subsets) and adds subsets one by one until every element in S has been covered; each time it chooses the subset covering the largest number of uncovered elements. It is well known that this greedy algorithm is a $1 + \ln|S|$ approximation algorithm for Minimum Set Cover.

LEMMA 1. *[13] Greedy Set Cover Algorithm outputs a set cover of size at most $1 + \ln|S|$ times the minimum set cover size.*

The *Greedy Minimum Key Algorithm* mimics the greedy set cover algorithm: start with an empty set of attributes and add attributes one by one until all tuple pairs are separated; each time chooses an attribute separating the largest number of tuple pairs not yet separated. The running time of the algorithm is $O(m^3 n)$. It is easy to infer the approximation ratio of this algorithm from Lemma 1:

THEOREM 2. *Greedy Minimum Key Algorithm outputs a key of size at most $1 + 2\ln n$ times the minimum key size.*

The greedy algorithms are optimal because neither Minimum Set Cover nor Minimum Test Collection is approximable within $c \ln|S|$ for some $c > 0$ [10]. Note that this is the worst case bound and in practice the algorithms usually find much smaller set covers or keys.

2.3 (ϵ, δ)-Separation Minimum Key

The greedy algorithm in the last section is optimal in terms of approximation ratio, however, it requires multiple scans ($O(m^2)$ scans indeed) of the table, which is expensive for large data sets. In this and next section, we relax the minimum key problem by allowing quasi-identifiers and design efficient algorithms with approximate guarantees.

We use the standard (ϵ, δ) formulation: with probability at least $1 - \delta$, we allow an "error" of at most ϵ, i.e. we output a quasi-identifier with separation (distinct) ratio at least $1-\epsilon$. The (ϵ, δ) Minimum Set Cover Problem is defined similarly and requires the output set cover covering at least a $1 - \epsilon$ fraction of all elements.

Our algorithms are based on random sampling. We first randomly sample k elements (tuples), and reduce the input set cover (key) instance to a smaller set cover (key) instance containing only the sampled elements (tuples). We then solve the exact minimum set cover (key) problem in the smaller instance (which is again a hard problem but has much smaller size, so we can afford to apply the greedy algorithms in Section 2.2), and output the solution as an approximate solution to the original problem. The number of samples k is carefully chosen so that the error probability is bounded. We present in detail the algorithm for (ϵ, δ)-set cover in Section 2.3.1; the (ϵ, δ)-Separation Minimum Key problem can be solved by reducing to (ϵ, δ) Minimum Set Cover (Section 2.3); we discuss (ϵ, δ)-Distinct Minimum Key in Section 2.4.

2.3.1 (ϵ, δ) Minimum Set Cover

The key observation underlying our algorithm is that to check whether a given collection of subsets is a set cover, we only need to check some randomly sampled elements if we allow approximate solutions. If the collection only covers part of S, then it will fail the check after enough random samples. The idea is formalized as the following lemma.

LEMMA 3. *$s_1, s_2, \ldots, s_k$ are k elements independently randomly chosen from S. If a subset S' satisfies $|S'| < \alpha|S|$, then $Pr[s_i \in S', \forall i] < \alpha^k$.*

The proof is straightforward. The probability that a random element of S belongs to S' is $|S'|/|S| < \alpha$, therefore the probability of all k random elements belonging to S' is at most α^k.

Now we combine the idea of random sample checking with the greedy algorithm for the exact set cover. Our *Greedy Approximate Set Cover algorithm* is as follows:

1. Choose k elements uniformly at random from S (k is defined later);
2. Reduce the problem to a smaller set cover instance: the ground set $\tilde{S}$ consists of the k chosen elements; each subset in the original problem maps to a subset which is the intersection of $\tilde{S}$ and the original subset;
3. Apply Greedy Set Cover Algorithm to find an exact set cover for $\tilde{S}$, and output the solution as an approximate set cover to S.

Let n be the size of the ground set S, and m be the number of subsets. We say a collection of subsets is an *α-set cover* if it covers at least a α fraction of the elements.

THEOREM 4. *With probability $1 - \delta$, the above algorithm with $k = \log_{\frac{1}{1-\epsilon}} \frac{2^m}{\delta}$ outputs a $(1 - \epsilon)$-set cover whose cardinality is at most $(1 + \ln\log_{\frac{1}{1-\epsilon}} \frac{2^m}{\delta})|I^*|$, where I^* is the optimal exact set cover.*

PROOF. Denote by $\tilde{S}$ the ground set of the reduced instance ($|\tilde{S}| = k$); by $\tilde{I}^*$ the minimum set cover of $\tilde{S}$. The greedy algorithm outputs a subcollection of subsets covering all k elements of $\tilde{S}$, denoted by $\tilde{I}$. By Lemma 1, $|\tilde{I}| \le (1 + \ln|\tilde{S}|)|\tilde{I}^*|$. Note that I^*, the minimum set cover of the original set S, corresponds to a set cover of $\tilde{S}$, so $|\tilde{I}^*| \le |I^*|$, and hence $|\tilde{I}| \le (1 + \ln k)|I^*|$.

We map $\tilde{I}$ back to a subcollection I of the original problem. We have
$|I| = |\tilde{I}| \le (1 + \ln k)|I^*| = (1 + \ln\log_{\frac{1}{1-\epsilon}} \frac{2^m}{\delta})|I^*|$.

Now bound the probability that I is not a $1 - \epsilon$-set cover. By Lemma 3, the probability that a subcollection covering less than a $1 - \epsilon$ fraction of S covers all k chosen elements of $\tilde{S}$ is at most

$$(1 - \epsilon)^k = (1 - \epsilon)^{\log_{\frac{1}{1-\epsilon}} \frac{2^m}{\delta}} = (1 - \epsilon)^{\log_{1-\epsilon} \frac{\delta}{2^m}} = \frac{\delta}{2^m}.$$

There are 2^m possible subcollections; by union bound, the overall error probability, i.e. the probability that any subcollection is not a $(1 - \epsilon)$-cover of S but is an exact cover of $\tilde{S}$, is at most δ. Hence, with probability at least $1 - \delta$, I is a $(1 - \epsilon)$-set cover for S. $\square$

If we take ϵ and δ as constants, the approximation ratio is essentially $\ln m + O(1)$, which is smaller than $1 + \ln n$ when $n \gg m$. The space requirement of the above algorithm is $mk = O(m^2)$ and running time is $O(m^4)$.

2.3.2 (ϵ, δ)-Separation Minimum Key

The reduction from Minimum Key to Minimum Set Cover preserves the separation ratio: an α-separation quasi-identifier separates at least an α fraction of all pairs of tuples, so its corresponding subcollection is an α-set cover; and vice versa. Therefore, we can reduce the (ϵ, δ)-Separation Minimum Key problem to the (ϵ, δ)-Set Cover problem where $|S| = O(n^2)$. The complete algorithm is as follows.

1. Randomly choose $k = \log_{\frac{1}{1-\epsilon}} \frac{2^m}{\delta}$ pairs of tuples;
2. Reduce the problem to a set cover instance where the ground set $\tilde{S}$ is the set of those k pairs and each attribute maps to a subset of the k pairs separated by this attribute;
3. Apply Greedy Set Cover Algorithm to find an exact set cover for $\tilde{S}$, and output the corresponding attributes as a quasi-identifier to the original table.

THEOREM 5. *With probability* $1-\delta$*, the above algorithm outputs a* $(1-\epsilon)$*-separation quasi-identifier whose size is at most* $(1+\ln\log_{\frac{1}{1-\epsilon}} \frac{2^m}{\delta})|I^*|$*, where* I^* *is the smallest key.*

The proof directly follows Theorem 4. The approximation ratio is essentially $\ln m + O(1)$. The space requirement of the above algorithm is $mk = O(m^2)$, which significantly improves upon the input size mn.

2.4 (ϵ, δ)-Distinct Minimum Key

Unfortunately, the reduction to set cover does not necessarily map an α-distinct quasi-identifier to an α-set cover. As pointed out in Section 1.1, an α-distinct quasi-identifier corresponds to an α'-separation quasi-identifier, and thus reduces to an α'-set cover, where α' can be as small as $2\alpha-\alpha^2$, or as large as $1-\frac{2(1-\alpha)}{n}$. Therefore reducing this problem directly to set cover gives too loose bound, and a new algorithm is desired.

Our algorithm for finding distinct quasi-identifiers is again based on random sampling. We reduce the input (ϵ, δ)-Distinct Minimum Key instance to a smaller (exact) Minimum Key instance by randomly choosing k tuples and keeping all m attributes. The following lemma bounds the probability that a subset of attributes is an (exact) key in the sample table, but not an α-distinct quasi-identifier in the original table.

LEMMA 6. *Randomly choose k tuples from input table T to form table T_1. Let p be the probability that an (exact) key of T_1 is not an α-distinct quasi-identifier in T. Then*

$$p < e^{-\frac{(\frac{1}{\alpha}-1)k^2}{2n}}$$

PROOF. Suppose we have n balls distributed in $d = \alpha n$ distinct bins. Randomly choose k balls without replacement, and the probability that the k balls are all from different bins is exactly p. Let $x_1, x_2, \ldots, x_d$ be the number of balls in the d bins ($\sum_{i=1}^{d} x_i = n, x_i > 0$), then

$$p = \frac{\sum_{all\{i_1,i_2,\ldots,i_k\}} x_{i_1}x_{i_2}\ldots x_{i_k}}{\binom{n}{k}}.$$

p is maximized when all x_is are equal, i.e. each bin has $\frac{1}{\alpha}$ balls. Next we compute p for this case. The first ball can be from any bin; to choose the second ball, we have $n-1$ choices, but it cannot be from the same bin as the first one, so $\frac{1}{\alpha}-1$ of the $n-1$ choices are infeasible; similar arguments hold for the remaining balls. Summing up, the probability that all k balls are from distinct bins is

$$\begin{aligned} p &= 1(1-\frac{\frac{1}{\alpha}-1}{n-1})(1-\frac{2(\frac{1}{\alpha}-1)}{n-2})\ldots(1-\frac{(k-1)(\frac{1}{\alpha}-1)}{n-(k-1)}) \\ &\le e^{-(\frac{\frac{1}{\alpha}-1}{n-1}+\frac{2(\frac{1}{\alpha}-1)}{n-2}+\frac{(k-1)(\frac{1}{\alpha}-1)}{n-(k-1)})} \\ &< e^{-\frac{(\frac{1}{\alpha}-1)k^2}{2n}} \end{aligned}$$

□

The Greedy (ϵ, δ)-Distinct Minimum Key Algorithm is as follows:

1. Randomly choose $k = \sqrt{\frac{2(1-\epsilon)}{\epsilon} n \ln \frac{2^m}{\delta}}$ tuples and keep all attributes to form table T_1;
2. Apply Greedy Minimum Key Algorithm to find an exact key in T_1, and output it as a quasi-identifier to the original table.

THEOREM 7. *With probability* $1-\delta$*, the above algorithm outputs a* $(1-\epsilon)$*-distinct quasi-identifier whose size is at most* $(1+2\ln k)|I^*|$*, where* I^* *is the smallest exact key.*

The proof is similar to Theorem 4, substituting Lemma 3 with Lemma 6. k is chosen such that $p \le \frac{\delta}{2^m}$ to guarantee that the overall error probability is less than δ. The approximation ratio is essentially $\ln m + \ln n + O(1)$, which improves the $1+2\ln n$ result for the exact key. The space requirement is $mk = O(m\sqrt{mn})$, sublinear in the number of tuples of the original table.

2.5 Minimum β-Separation Quasi-identifier

In previous sections, our goal is to find a small quasi-identifier that is almost a key. Note that ϵ indicates our "error tolerance", not our goal. For (ϵ, δ)-Separation Minimum Key problem, our algorithm is likely to output quasi-identifiers whose separation ratios are far greater than $1-\epsilon$. For example, suppose the minimum key of a given table consists of 100 attributes, while the minimum 0.9-separation quasi-identifier has 10 attributes, then our $(0.1, 0.01)$-separation algorithm may output a quasi-identifier that has say 98 attributes and is 0.999-separation. However, sometimes we may be interested in finding 0.9-separation quasi-identifiers which have much smaller sizes. For this purpose we consider the *Minimum β-Separation Quasi-identifier Problem*: find a quasi-identifier with the minimum size and separation ratio at least β.

The Minimum β-Separation Quasi-identifier Problem is even harder than Minimum Key as the latter is a special case where $\beta = 1$. So again we consider the approximate version by relaxing the separation ratio: we require the algorithm to output a quasi-identifier with separation ratio at least $(1-\epsilon)\beta$ with probability at least $1-\delta$.

We present the algorithm for approximate β-set cover; the β-separation quasi-identifier problem can be reduced to β-set cover as before.

The *Greedy Minimum β-Set Cover algorithm* works as follows: first randomly sample $k = \frac{16}{\beta\epsilon^2} \ln \frac{2^m}{\delta}$ elements from the ground set S, and construct a smaller set cover instance defined on the k chosen elements; run the greedy algorithm on the smaller set cover instance until get a subcollection covering at least $(2-\epsilon)\beta k/2$ elements (start with an empty subcollection; each time add to the subcollection a subset covering the largest number of uncovered elements).

THEOREM 8. *The Greedy Minimum β-Set Cover algorithm runs in space* $mk = O(m^2)$*, and with probability at least* $1-\delta$*, outputs a* $(1-\epsilon)\beta$*-set cover with size at most* $(1+\ln\frac{(2-\epsilon)\beta k}{2})|I^*|$*, where* I^* *is the minimum β-set cover of S.*

The proof can be found in the appendix. This algorithm also applies to the minimum exact set cover problem (the special case where $\beta = 1$), but the bound is worse than Theorem 4; see the appendix for more detailed comparison.

The minimum β-separation quasi-identifier problem can be solved by reducing to β-set cover problem and applying the above greedy algorithm. Unfortunately, we cannot provide similar algorithms for β-distinct quasi-identifiers; the main difficulty is that it is hard to give a tight bound to the distinct ratio of the original table by only looking at a small sample of tuples. The negative results on distinct ratio estimation can be found in [6].

3. FINDING ALL MINIMAL KEYS AND QUASI-IDENTIFIERS

In this section, we consider the problem of finding all minimal keys, β-separation quasi-identifiers, and β-distinct quasi-identifiers. This problem is inherently hard as the number of minimal keys can be exponential in the number of attributes m, so it is inevitable that the worst case running time is exponential in m. Nevertheless we want to design algorithms efficient in practice, at least for tables with a small number of attributes.

All algorithms in this section perform a search in the lattice of attribute subsets (we describe the basic search procedure in Section 3.1). The key idea for improvement is to use small sample tables to detect non-keys quickly. In order not to prune any key (quasi-identifier) by mistake, we need to find necessary conditions in the sample table for an attribute set to be a key (quasi-identifier) in the entire table. The necessary conditions are different for keys, separation, and distinct quasi-identifiers, and are addressed in Section 3.1, 3.2 and 3.3 respectively.

3.1 Finding All Minimal Exact Keys

We first describe the brute-force levelwise algorithm to find all exact keys. Then we will improve upon the basic algorithm by pruning with random samples.

The collection of all possible attribute subsets form a set containment lattice, the bottom of which are all singleton sets and the top is the set of all attributes. Since we are only concerned with the minimal keys, once find a key, we discard all its superset in the lattice. Levelwise algorithm [18] is an efficient algorithm to perform such search on lattices and has been exploited in many data mining applications. It starts the search from the singleton sets and works its way up to the top of the lattice. Once a key is found, all its supersets are pruned from the lattice and never checked later. Please refer to [11, 12] for more detail on levelwise algorithm.

Now we improve upon the brute-force algorithm by introducing techniques to effectively prune the lattice. We make use of the following simple fact.

Fact *A key of the entire table is still a key in any sub-table; or equivalently, if a attribute set is not a key in some sub-table, it cannot be a key for the entire table.*

Our pruning algorithm first samples some random tuples to form a small table and find all minimal keys in the small table, which defines a lower border in the lattice. Then we use the levelwise algorithm to search keys in the original table, but start from the lower border instead of the bottom of lattice. [1]

Furthermore, we can apply the pruning idea iteratively by constructing a series of tables with increasing tuple numbers. The minimal keys of table i defines a lower border in the lattice where we will start the search for the minimal keys of table $i+1$.

We implemented all three algorithms (brute-force, pruning with one sub-table, iterative pruning). To deal with large tables that cannot fit in memory, we keep the table in database and check if an attribute set is a key by issuing the SQL query "select count(*) from (select distinct <list of attributes in the checked set> from table) " and comparing the count with the total tuple number. Experiments show that pruning with small sub-tables improves the running time by orders of magnitude, especially for large tables.

3.2 Finding All Minimal β-Separation Quasi-identifiers

The brute-force algorithm for all minimal β-separation quasi-identifiers is similar to that of exact keys: use the levelwise algorithm to search the lattice except that now we check whether an attribute set is a β-separation quasi-identifier.

However, pruning with sub-tables for quasi-identifiers is not as straightforward as exact keys because the Fact in Section 3.1 does not extend trivially to quasi-identifiers, i.e. a β-separation quasi-identifier in the entire table is not necessarily β-separation in a sub-table. Fortunately, we have the following lemma analogous to the Fact. The proof of the lemma is similar to the proof of Lemma 13 in the appendix.

LEMMA 9. *Randomly sample k pairs of tuples, then a β-separation quasi-identifier separates at least $\alpha\beta$ of the k pairs, with probability at least $1-e^{-(1-\alpha)^2\beta k/2}$.*

When prune with a sub-table of size k, to guarantee that with probability $1-\delta$, no β-separation quasi-identifier is mistakenly pruned, we can set the parameter $\alpha = 1-\sqrt{\frac{2\ln(2^m/\delta)}{\beta k}}$, and prune all attribute sets with separation ratio less than $\alpha\beta$ in the sub-table. Again we can construct a series of tables and conduct pruning iteratively.

3.3 All Minimal β-Distinct Quasi-identifiers

The idea of pruning with sub-tables is also applicable to finding all β-distinct keys. We can use the following lemma to prune in sub-tables.

LEMMA 10. *Randomly sample k tuples from the input table T into a small table T_1 ($k \ll n$, where n is the number of tuples in T). A β-distinct quasi-identifier of T is an $\alpha\beta$-distinct quasi-identifier of T_1 with probability at least $1-e^{-(1-\alpha)^2\beta k/2}$.*

PROOF. By the definition of β-distinct quasi-identifier, the tuples has at least βn distinct values projected on the quasi-identifier. Take (any) one tuple from each distinct value, and call those representing tuples "good tuples". There are at least βn good tuples in T.

Let k_1 be the number of distinct values in T_1 projected on the quasi-identifier, and k' be the number of good tuples

[1]There are two obvious alternatives. One is that when check if an attribute set is a key, keep fetching the next tuple until detect a collision. However the check is hard to implement efficiently if the table cannot be fit in memory. The second alternative is that when check if an attribute set is a key, first check in a small sub-table, and check the original table only if it is a key in sub-table. We implemented this method and found the performance is worse, probably due to the redundance of checking sub-table once above the lower border defined by the minimal keys of the sub-table.

in T_1. We have $k_1 \geq k'$ because all good tuples are distinct. (The probability that any good tuple is chosen more than once is negligible when $k \ll n$.) Next we bound the probability $Pr[k' \leq \alpha\beta k]$. Since each random tuple has a probability at least β of being good, and each sample are chosen independently, we can use Chernoff bound (see [21] Ch. 4) and get

$$Pr[k' \leq \alpha\beta k] \leq e^{-(1-\alpha)^2\beta k/2}$$

Since $k_1 \geq k'$, we have

$$Pr[k_1 \leq \alpha\beta k] \leq Pr[k' \leq \alpha\beta k] \leq e^{-(1-\alpha)^2\beta k/2}$$

Hence with probability at least $1 - e^{-(1-\alpha)^2\beta k/2}$, the quasi-identifier has distinct ratio at least $\alpha\beta$ in T_1. □

When prune with a sub-table of size k, to guarantee that with probability $1-\delta$ no β-distinct quasi-identifier is mistakenly pruned, we can set the parameter $\alpha = 1 - \sqrt{\frac{2\ln(2^m/\delta)}{\beta k}}$, and prune all attribute sets with distinct ratio less than $\alpha\beta$ in the sub-table.

4. MASKING QUASI-IDENTIFIERS

In this section we consider the quasi-identifier masking problem: when we release a table, we want to publish a subset of the attributes subject to the privacy constraint that no β-separation (or β-distinct) quasi-identifier is published; on the other hand we want to maximize the utility, which is measured by the number of published attributes. For each problem, we first present a greedy algorithm which generates good results but runs slow for large tables, and then show how to accelerate the algorithms using random sampling. (The algorithms can be easily extended to the case where the attributes have weights and the utility is the sum of attribute weights.)

4.1 Masking β-Separation Quasi-identifiers

As in Section 2.2, we can reduce the problem to a set cover type problem: let the ground set S be the set of all pairs of tuples, and let each attribute correspond to a subset of tuple pairs separated by this attribute, then the problem of Masking β-Separation Quasi-identifier is equivalent to finding a maximum number of subsets such that at most a β fraction of elements in S is covered by the selected subsets. We refer to this problem as *Maximum Non-Set Cover problem*. Unfortunately, the Maximum Non-Set Cover problem is NP-hard by a reduction from the Dense Subgraph problem. (See the appendix for the hardness proof.)

We propose a greedy heuristic for masking β-separation quasi-identifiers: start with an empty set of attributes, and add attributes to the set one by one as long as the separation ratio is below β; each time pick the attribute separating the least number of tuple pairs not yet separated.

The algorithm produces a subset of attributes satisfying the privacy constraint and with good utility in practice, however it suffers from the same efficiency issue as the greedy algorithm in Section 2.2: it requires $O(m^2)$ scans of the table and is thus slow for large data sets. We again use random sampling technique to accelerate the algorithm: Lemma 9 already gives a necessary condition for a β-separation quasi-identifier in the sample table (with high probability), so only looking at the sample table and pruning all attribute sets satisfying the necessary condition will guarantee the privacy constraint.

The *Greedy Approximate β-Separation Masking Algorithm* is as follows:

1. Randomly choose k pairs of tuples;
2. Let $\beta' = (1 - \sqrt{\frac{2\ln(2^m/\delta)}{\beta k}})\beta$. Run the following greedy algorithm on the selected pairs: start with an empty set C of attributes, and add attributes to the set C one by one as long as the number of separated pairs is below $\beta' k$; each time pick the attribute separating the least number of tuple pairs not yet separated;
3. Publish the set of attributes C.

By the nature of the algorithm the published attributes C do not contain quasi-identifiers with separation greater than β' in the sample pairs; by Lemma 9, this ensures that with probability at least $1 - 2^m e^{-(1-\beta'/\beta)^2\beta k/2} = 1 - \delta$, C do not contain any β-separation quasi-identifier in the original table. Therefore the attributes published by the above algorithm satisfies the privacy constraint.

THEOREM 11. *With probability at least $1 - \delta$, the above algorithm outputs an attribute set with separation ratio at most β.*

We may over-prune because the condition in Lemma 9 is not a sufficient condition, which means we may lose some utility. The parameter k in the algorithm offers a tradeoff between the time/space complexity and the utility. Obviously both the running time and the space increase linearly with k; on the other hand, the utility (the number of published attributes) also increases with k because the pruning condition becomes tighter as k increases. Our experiment results show that the algorithm is able to dramatically reduce the running time and space complexity, without much sacrifice in the utility (see Section 5).

4.2 Masking β-Distinct Quasi-identifiers

For masking β-distinct quasi-identifiers, we can use a similar greedy heuristic: start with an empty set of attributes, and each time pick the attribute adding the least number of distinct values, as long as the distinct ratio is below β. And similarly we can use a sample table to trade off utility for efficiency.

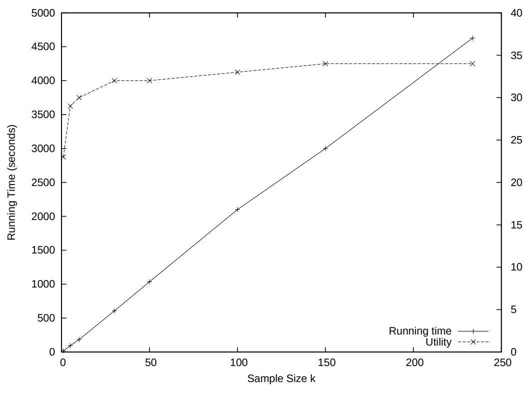
1. Randomly choose k tuples and keep all the columns to form a sample table T_1;
2. Let $\beta' = (1 - \sqrt{\frac{2\ln(2^m/\delta)}{\beta k}})\beta$. Run the following greedy algorithm on T_1: start with an empty set C of attributes, and add attributes to the set C one by one as long as the distinct ratio is below β'; each time pick the attribute adding the least number of distinct values;
3. Publish the set of attributes C.

The following theorem states the privacy guarantee of the above algorithm, which follows easily from Lemma 10.

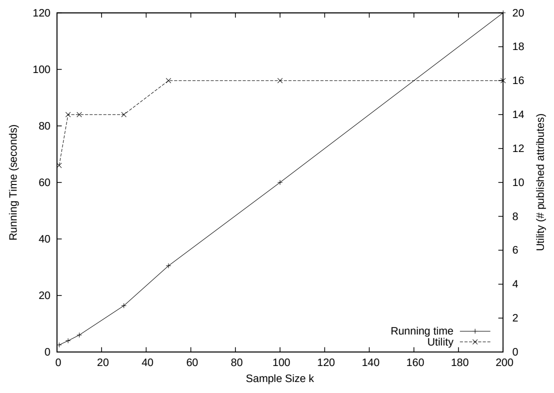
THEOREM 12. *With probability at least $1-\delta$, the attribute set published by the algorithm has distinct ratio at most β.*

5. EXPERIMENTS

We have implemented all algorithms for finding and masking quasi-identifiers, and conducted extensive experiments using real data sets. All experiments were run on a 2.4GHz Pentium PC with 1GB memory.



(a) Masking 0.5-distinct quasi-identifiers



(b) Masking 0.8-separation quasi-identifiers

Figure 1: Performance of masking quasi-identifier algorithms with different sample sizes on *california*. Figures (a) and (b) show how the running time (the left y axis) and the utility (the right y axis) changes with the sample size (the parameter k) in Greedy Approximate algorithms for masking 0.5-distinct and 0.8-separation quasi-identifiers respectively. Note the unit of x axis is 1000 tuples/pairs.

5.1 Data Sets

One source of data sets is the census microdata "Public-Use Microdata Samples (PUMS)" [1], provided by US Census Bureau. We gather the 5 percent samples of Census 2000 data from all states and put into a table "census". To study the performance of our algorithms on tables with different sizes, we also extract 1 percent samples of state-level data and select 4 states with different population sizes – Idaho, Washington, Texas and California. We extract 41 attributes including age, sex, race, education level, salary etc. We only use adult records (age ≥ 20) because many children are indistinguishable even with all 41 attributes. The table *census* has 10 million distinct adults, and the sizes of *Idaho, Washington, Texas* and *California* are 8867, 41784, 141130 and 233687 respectively.

We also use two data sets *adult* and *covtype* provided by UCI Machine Learning Repository [23]. The *covtype* table has 581012 rows and 54 attributes. We use 14 attributes of *adult* including age, education level, marital status; the number of records in *adult* is around 30000.

5.2 Masking Quasi-identifiers

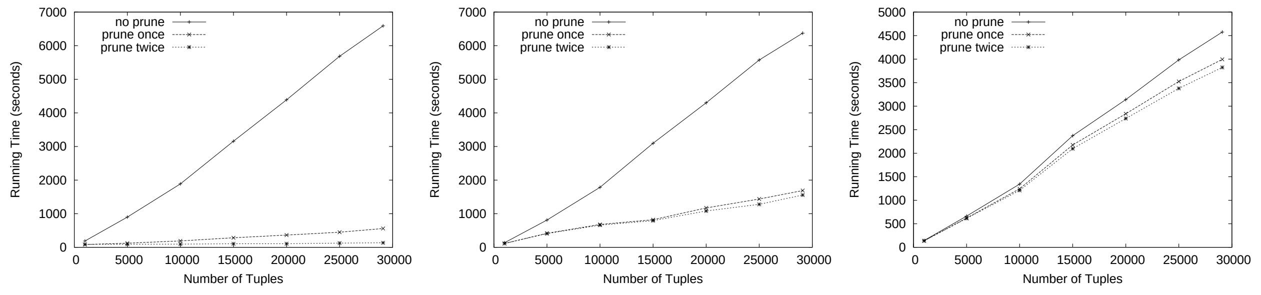
The greedy approximate algorithms for masking quasi-identifiers are randomized algorithms that guarantee to satisfy the privacy constraints with probability $1 - \delta$. We set $\delta = 0.01$, and the privacy constraint are satisfied in all experiments, which confirms the accuracy of our algorithms.

Figure 1 shows the tradeoff between the running time and the utility (the number of attributes published), using the *california* data set. Both the running time and the utility decrease as the sample size k decreases; however, the running time decreases linearly with k while the utility degrades very slowly. For example, running the greedy algorithm for masking 0.5-distinct quasi-identifiers on the entire table (without random sampling) takes 80 minutes and publishes 34 attributes (the rightmost point in Figure a); using a sample of 30000 tuples the greedy algorithm takes only 10 minutes and outputs 32 attributes. The impact of k on the utility for masking separation quasi-identifier is even minor (Figure b). To run the greedy algorithm for masking 0.8-separation quasi-identifier on the entire table takes 728 seconds (not shown in the figure); using a sample of 50000 pairs offers the same utility and only takes 30 seconds. The results show that our random sampling technique can greatly improve time and space complexity (space is also linear in k), with only minor sacrifice on the utility.

Data Sets	Greedy		Greedy Approximate	
	time	utility	time	utility
adult	36s	12	-	-
covtype	-	-	2000s	46
idaho	172s	33	-	-
wa	880s	34	620s	33
texas	3017s	35	630s	33
ca	4628s	34	606s	32
census	-	-	755s	30

Table 2: Algorithms for masking 0.5-distinct quasi-identifiers. The column "Greedy" represents the greedy algorithm on the entire table, and the column "Greedy Approximate" represents running greedy algorithm on a random sample of 30000 tuples. We compare the running time and the utility (the number of published attributes) of the two algorithms on different data sets. The results of Greedy on *census* and *covtype* are not available because the algorithm does not terminate in 10 hours; the results of Greedy Approximate on *adult* and *Idaho* are not available because the input tuple number is less than 30000.

Table 2 and 3 compare the running time and the utility (the number of published attributes) of running the greedy algorithm on the entire table versus on a random sample (we use a sample of 30000 tuples in Table 2 and a sample of 50000 pairs of tuples in Table 3). Results on all data sets confirm that the random sampling technique is able



(a) All exact keys (b) All 0.9-Distinct Q.ID.s (c) All 0.999-Separation Q.ID.s

Figure 2: Comparison of running time of finding all minimal quasi-identifiers using different pruning methods. The three figures show the running time of three algorithms for finding all keys, 0.9-distinct and 0.999-separation quasi-identifiers on *adult* table respectively. The "pruning once" algorithm uses a 10% sub-table; the iterative pruning algorithm uses two sub-tables, the first one with 1% samples and the second one 10%.

Data Sets	Greedy		Greedy Approximate	
	time	utility	time	utility
adult	19s	5	2s	5
covtype	2 hours	38	104s	37
idaho	147s	24	30s	23
wa	646s	23	35s	23
texas	1149s	19	34s	19
ca	728s	16	30s	16
census	-	-	170s	17

Table 3: Algorithms for masking 0.8**-separation quasi-identifiers.** The column "Greedy" represents the greedy algorithm on the entire table, and the column "Greedy Approximate" represents running greedy algorithm on a random sample of 50000 pairs of tuples. We compare the running time and the utility of the two algorithms on different data sets. The result of Greedy on *census* is unavailable because the algorithm does not terminate in 10 hours.

to reduce the running time dramatically especially for large tables, with only minor impact on the utility. For the largest data set *census*, running the greedy algorithm on the entire table does not terminate in 10 hours, while with random sampling it only takes no more than 13 minutes for masking 0.5-distinct quasi-identifier and 3 minutes for masking 0.8-separation quasi-identifier.

5.3 Finding All Minimal Quasi-identifiers

We implement algorithms for finding all minimal keys, β-distinct and β-separation quasi-identifiers. For each of them, we implement 3 algorithms: no pruning, pruning with one sub-table, iterative pruning.

The pruning algorithms for distinct and separation quasi-identifiers are randomized algorithms that guarantee to output the correct result with probability $1 - \delta$. In all experiments we set $\delta = 0.01$, and the algorithms are always able to find all minimal quasi-identifiers correctly throughout the experiments, which confirms the accuracy of our algorithms.

We measure the running time of the three algorithms on *adult* table, and the results are illustrated in Figure 2. The figures show that pruning is highly effective for minimal keys and distinct quasi-identifiers, while the improvement for separation quasi-identifiers is marginal; iterative pruning is most effective for exact keys. For example, to find all minimal keys, it takes 6590 seconds without pruning, 559 seconds if prune with one sub-table, and only 134 seconds if prune with two sub-tables; to find all 0.9-distinct quasi-identifiers, pruning improves the running time from 6373 seconds to around 1600 seconds. The improvement on separation quasi-identifiers are not significant, because there are a large number of small separation quasi-identifiers causing pruning ineffective in reducing the search space. We also generate input tables of different sizes by taking random samples from the original table; the running time of all algorithms increases linearly with the number of input tuples.

We next study the influence of pruning levels and sub-table sizes on the running time. From Figure 3, we can see that iterative pruning outperforms pruning with one sub-table in most cases, especially when the pruning table size is large; using three levels of pruning is slightly better than using two levels. Fixing the pruning level and increasing the sub-table sizes, the running time first decreases because larger sub-tables are more effective in pruning the search space, and then increases after a certain point due to the large cost of sampling and processing the sub-tables. We do not have theoretical results for the optimal pruning level and table sizes; they depend on the characteristics of the input table, such as the sizes and numbers of the quasi-identifiers. Our simulation on various tables show that two or three levels of pruning with each table size 10% of the next level usually provides reasonably good performance.

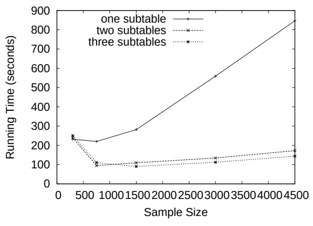
Even though our algorithms have effectively reduced the running time, they are yet unable to process larger data sets as the running time is still exponential in the number of attributes.

5.4 Approximate Minimum Key Algorithms

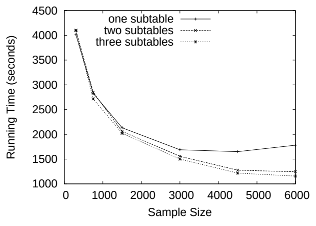
Finally we examine the greedy algorithms for finding minimum key and (ϵ, δ)-separation or -distinct minimum key in Section 2. Table 4 shows the experimental results of the Greedy Minimum Key, Greedy $(0.1, 0.01)$-Distinct Minimum

Data Sets	Greedy		distinct Greedy ($\epsilon = 0.1$)			separation Greedy ($\epsilon = 0.001$)		
	time	key size	time	key size	distinct ratio	time	key size	separation ratio
adult	35.5s	13	8.8s	13	1.0	3.11s	5	0.99995
covtype	964s	5	78.1s	3	0.9997	27.1s	2	0.999996
idaho	50.4s	14	15.2s	8	0.997	1.07s	3	0.9999
wa	490s	22	34.1s	8	0.995	7.14s	3	0.99993
texas	2032s	29	120s	14	0.995	13.2s	4	0.99995
ca	3307s	29	145s	13	0.994	16.3s	4	0.99998
census	-	-	808s	17	0.993	120s	3	0.99998

Table 4: Running time and output key sizes of the Greedy Minimum Key, Greedy $(0.1, 0.01)$**-Distinct Minimum Key, and Greedy** $(0.001, 0.01)$**-Separation Minimum Key algorithms.** The result of Greedy Minimum Key on *census* is not available because the algorithm does not terminate in 10 hours.



(a) All exact keys



(b) All 0.9-distinct Q.ID.

Figure 3: Running time using different pruning levels and sub-table sizes. Figures (a) and (b) show the running time (y axis) for finding all keys and 0.9-distinct quasi-identifiers in *adult* table. The three curves use one, two, three pruning tables respectively. The x axis gives the pruning table size at the last level; if prune with more than one table, the table size at each level is 10% of the next level, or 100 tuples at minimum.

Key, and Greedy $(0.001, 0.01)$-Separation Minimum Key algorithms on different data sets.

The Greedy Minimum Key algorithm (applying greedy algorithm directly on the entire table) works well for small data sets such as *adult, idaho*, but becomes unaffordable as the data size increases. The approximate algorithms for separation or distinct minimum key are much faster. For the table *California*, the greedy minimum key algorithm takes almost one hour, while the greedy distinct algorithm takes 2.5 minutes, and greedy separation algorithm merely seconds; for the largest table *census*, the greedy minimum key algorithm takes more than 10 hours, while the approximate algorithms take no more than 15 minutes. The space and time requirements of our approximate minimum key algorithms are sublinear in the number of input tuples, and we expect the algorithms to scale well on even larger data sets.

We measure the distinct and separation ratios of the output quasi-identifiers, and find the ratios always within error ϵ. This confirms the accuracy of our algorithms.

Theorem 5 and 7 provide the theoretical bounds on the size of the quasi-identifiers found by our algorithms ($\ln m$ or $\ln mn$ times the minimum key size). Those bounds are worst case bounds, and in practice we usually get much smaller quasi-identifiers. For example, the minimum key size of *adult* is 13, and the greedy algorithm for both distinct and separation minimum key find quasi-identifiers no larger than the minimum key. (For other data sets in Table 4, computing the minimum key exactly takes prohibitively long time, so we are not able to verify the approximation ratio of our algorithms.) We also generate synthetic tables with known minimum key sizes, then apply the greedy distinct minimum key algorithm (with $\epsilon = 0.1$) on those tables and are always able to find quasi-identifiers no larger than the minimum key size. Those experiments show that in practice our approximate minimum key algorithms usually perform much better than the theoretical worst case bounds, and are often able to find quasi-identifiers with high separation (distinct) ratio and size close to the minimum key.

6. RELATED WORK

The implication of quasi-identifiers to privacy is first formally studied by Sweeney, who also proposed the k-anonymity framework as a solution to this problem [27, 26]. Afterwards there is numerous work which studies the complexity of this problem [19, 2], designs and implements algorithms to achieve k-anonymity [25, 5], or extends upon the framework [17, 15]. Our algorithm for masking quasi-identifiers can be viewed as an approximation to k-anonymity where the suppression must be conducted at the attribute level. Also it is an "on average" k-anonymity because it does not provide perfect anonymity for every individual but does so for the majority; a similar idea is used in [16]. On the other side, our algorithms for finding keys/quasi-identifiers attempt to attack the privacy of published data from the adversary's point of view, when the publish data is not k-anonymized. To the best of our knowledge, there is no existing work addressing this problem.

Our algorithms exploit the idea of using random samples to trade off between accuracy and space complexity, and can be viewed as streaming algorithms. Streaming algorithms emerged as a hot research topic in the last decade; see [3, 8, 22] for some examples. Our algorithms are similar to all these algorithms in the sense that we only need one scan of the input and space sublinear in the input data size, at the cost of only providing approximate answers. The novelty of our algorithms is that most classic streaming algorithms deal with a stream of values, whereas we deal with a stream of tuples with attribute structure.

The Minimum Key problem is closely related to the classic Minimum Set Cover and Minimum Test Collection problem, whose hardness and approximability are well studied in theoretical computer science field [9, 20, 10]. Although there has been extensive work on set cover problem, to the best of our knowledge, this paper is the first to study the trade-off between set coverage and space efficiency.

Keys are special cases of functional dependencies, and quasi-identifiers are a special case of approximate functional dependency. Our definitions of separation and distinct ratios for quasi-identifiers are adapted from the measures for quantifying approximations of functional dependencies proposed in [14, 24].

7. CONCLUSIONS AND FUTURE WORK

In this paper, we designed efficient algorithms for discovering and masking quasi-identifiers in large tables. We developed efficient algorithms that find small quasi-identifiers with provable size and separation/distinct ratio guarantees, with space and time complexity sublinear in the number of input tuples. We also designed efficient algorithms for finding all minimal quasi-identifiers, and for masking quasi-identifiers in large tables.

Most of our algorithms can be extended to the weighted case, where each attribute is associated with a weight and the size/utility of a set of attributes is defined as the sum of their weights. The idea of using random samples to trade off between accuracy and space complexity can potentially be explored in other problems on large tables.

8. REFERENCES

[1] Public-use microdata samples (pums). *http://www.census.gov/main/www/pums.html.*

[2] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D.Thomas, and A. Zhu. Anonymizing tables. In *ICDT*, 2005.

[3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, 1996.

[4] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, 2002.

[5] R. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, 2005.

[6] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards estimation error guarantees for distinct values. In *PODS*, 2000.

[7] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, 2003.

[8] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *SODA*, 2002.

[9] M. R. Garey and D. S. Johnson. Computers and intractability. 1979.

[10] B. V. Halldorsson, M. M. Halldorsson, and R. Ravi. Approximability of the minimum test collection problem. In *ESA*, 2001.

[11] H. H.Mannila and A. Verkamo. Discovery of frequent episodes in event sequences. In *Data Mining and Knowledge Discovery*, 1997.

[12] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Discovery of functional and approximate dependencies using partitions. In *ICDE*, 1998.

[13] D. Johnson. Approximation algorithms for combinatorial problems. In *J. Comput. System Sci.*, 1974.

[14] J. Kivinen and H. Mannila. Approximate dependency inference from relations. In *Theoretical Computer Science*, 1995.

[15] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, 2007.

[16] S. Lodha and D. Thomas. Probabilistic anonymity. *Technical Report.*

[17] Machanavajjhala, J. Gehrke, and D. Kifer. l-diversity: privacy beyond k-anonymity. In *ICDE*, 2006.

[18] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. In *Data Mining and Knowledge Discovery*, 1997.

[19] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *PODS*, 2004.

[20] B. Moret and H. Shapiro. On minimizing a set of tests. In *SIAM Journal on Scientific and Statistical Computing*, 1985.

[21] R. Motwani and P. Raghavan. Randomized algorithm. 1995.

[22] S. Muthukrishnan. Data streams: Algorithms and applications. 2005.

[23] D. Newman, S. Hettich, C. Blake, and C. Merz. Uci repository of machine learning databases. *http://www.ics.uci.edu/∼mlearn/MLRepository.html.*

[24] B. Pfahringer and S. Kramer. Compression-based evaluation of partial determinations. In *SIGKDD*, 1995.

[25] P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information. In *PODS*, 1998.

[26] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. In *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 2002.

[27] L. Sweeney. k-anonymity: a model for protecting privacy. In *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 2002.

9. APPENDIX

9.1 Minimum β-Set Cover

As the greedy approximate set cover algorithm in Section 2.3.1, we use random sampling to reduce the problem to a smaller set cover instance. We need enough samples to tell, with high probability, whether a subcollection covers more than β or less than $\alpha\beta$ fraction of the ground set.

LEMMA 13. *If we choose k elements from the ground set S, then for any given set S', we can tell whether $|S'| \le \alpha\beta|S|$ or $|S'| \ge \beta|S|$ with probability at least $1 - e^{-\frac{\beta k(1-\alpha)^2}{16}}$.*

PROOF. Let k' be the number of elements covered by S' in the k chosen elements.

If $|S'| \ge \beta|S|$, then a random element of S is covered by S' with probability at least β. Let x_i be an indicator random variable which is set to 1 if the ith chosen element is covered by S', and 0 otherwise. It is easy to see

$E[x_i] = Pr[x_i = 1] = |S'|/|S| \ge \beta$, and

$E[k'] = \sum_{i=1}^{k} E[x_i] \ge \beta k$.

Since all x_is are independent, we can apply Chernoff bound (see for example [21] Ch. 4),

$Pr[k' \le (1+\alpha)\beta k/2] < e^{-\frac{\beta k(1-\alpha)^2}{8}}$.

On the other hand, if $|S'| \le \alpha\beta|S|$, similarly we have

$Pr[k' \ge (1+\alpha)\beta k/2] < e^{-\frac{\beta k(1-\alpha)^2}{16}}$.

Therefore, by checking if S' covers more than $(1+\alpha)\beta/2$ fraction of the chosen elements, we can tell with high probability if S' covers at least β or at most $\alpha\beta$ fraction of S. $\square$

Note that Lemma 3 can be viewed as a special case of Lemma 13 where $\beta = 1$, but Lemma 3 provides a tighter bound for the special case. Suppose we want to tell whether one subcollection is an exact cover or not an α-cover with error probability at most δ. We need $k = \log_\alpha \delta$ samples according to Lemma 3, while Lemma 13 asks for $\frac{16}{(1-\alpha)^2} \ln \frac{1}{\delta}$ samples. For example, when $\alpha = 0.9$, $\log_\alpha \delta \approx 10 \ln \frac{1}{\delta}$, while $\frac{16}{(1-\alpha)^2} \ln \frac{1}{\delta} \approx 1600 \ln \frac{1}{\delta}$.

The *Greedy Minimum β-Set Cover algorithm* works as follows: first randomly sample $k = \frac{16}{\beta\epsilon^2} \ln \frac{2^m}{\delta}$ elements from the ground set S, and construct a smaller set cover instance defined on the k chosen elements; run the greedy algorithm on the smaller set cover instance until get a subcollection covering at least $(2-\epsilon)\beta k/2$ elements (start with an empty subcollection; each time add to the subcollection a subset covering the largest number of uncovered elements).

To prove the approximation ratio of the algorithm, we need the following lemma about approximation ratio of the greedy algorithm on partial set covers. The proof is similar with the original proof for the exact set cover.

LEMMA 14. *Apply the greedy algorithm for minimum set cover problem until get a γ-set cover. The size of result subcollection is within $1 + \ln \gamma n$ of the size of the minimum γ-set cover.*

PROOF. For each element e, define $price(e) = \frac{1}{size(c)}$, where c is the first subset covering e in the greedy algorithm, and $size(c)$ is the number of elements first covered by subset c in the greedy algorithm.

Number the elements of S in the order of which they were covered by the greedy algorithm, breaking ties arbitrarily. Let $e_1, \ldots, e_{\gamma n}$ be the numbering. Right before e_k is covered, at most $k-1$ elements have been covered, so OPT_γ covers at least $\gamma n - (k-1)$ uncovered elements. Since the greedy algorithm picks a subset c with the maximum $size(c)$, it follows that

$$price(e_k) \le \frac{OPT_\gamma}{\gamma n - (k-1)}$$

The size of γ-set cover output by the greedy algorithm equals to

$$\sum_{k=1}^{\gamma n} e_k = OPT_\gamma\left(\frac{1}{\gamma n} + \frac{1}{\gamma n - 1} + \ldots + 1\right) = (1 + \ln \gamma n) OPT_\gamma$$

$\square$

Now we are ready to prove Theorem 8. It is easy to check that the algorithm takes space $mk = \frac{16m}{\beta\epsilon^2} \ln \frac{2^m}{\delta}$.

Theorem 8 *With probability at least $1-\delta$, the Greedy Minimum β-Set Cover algorithm outputs a partial set cover which covers at least $(1-\epsilon)\beta$ of the ground set, and has size at most $(1 + \ln \frac{(2-\epsilon)\beta k}{2})|I^*|$, where I^* is minimum β-set cover of S.*

PROOF. We say a subcollection of subsets "good" if it covers at least $(2-\epsilon)\beta k/2$ of the k chosen elements.

We first bound the error probability that the algorithm outputs a subcollection covering less than $(1-\epsilon)\beta$ of S. According to Lemma 13, the probability that any such subcollection is good is less than

$$e^{-\frac{\beta k \epsilon^2}{16}} = e^{-\ln \frac{2^m}{\delta}} = \frac{\delta}{2^m}.$$

Suppose there are x such small subcollections, then with probability at least $1 - \frac{x\delta}{2^m}$ none of them is good.

Similarly, any β-cover of S will be good with probability $\frac{\delta}{2^m}$. Suppose there are y β-covers, then with probability at least $1 - \frac{y\delta}{2^m}$ all β-covers are good. Under the condition that all β-covers are good, it holds that $|I^*| \ge |\tilde{I^*}|$, where $\tilde{I^*}$ is the minimum good subcollection.

By Lemma 14, the greedy algorithm outputs a good subcollection whose size is within $1 + \ln \frac{(2-\epsilon)\beta k}{2}$ of the minimum good subcollection size $|\tilde{I^*}|$, thus also within $(1 + \ln \frac{(2-\epsilon)\beta k}{2})|I^*|$ under the condition that all β-covers are good.

The overall error probability is at most $\frac{(x+y)\delta}{2^m}$. Since $x+y$ is at most the total number of subset 2^m, the error probability is at most δ. $\square$

9.2 Max Non-Set Cover is NP-hard

THEOREM 15. *Maximum Non-Set Cover problem is NP-hard.*

PROOF. We reduce the Dense Subgraph problem to Maximum Non-Set Cover.

> *Dense Subgraph Problem:* given a graph $G = (V, E)$, find a subset of vertices T with cardinality at most k and the subgraph induced by T has the maximum number of edges.

Given a dense subgraph instance, we can reduce it to a max non-set cover instance: let the ground set be the set of vertices V; each edge corresponds to a subset of size 2 containing the two endpoints, then a subgraph of G with at most k vertices corresponds to a collection of subsets covering at most k elements, and vice versa. The dense subgraph problem is NP-hard by a reduction from Maximum Clique, therefore maximum non-set cover is also NP-hard. $\square$