

**A Theory and Toolkit for the Mathematics of Privacy:  
Methods for Anonymizing Data while Minimizing Information Loss**

by

Hooman Katirai  
BASc., Computer Engineering (2000)  
University of Waterloo

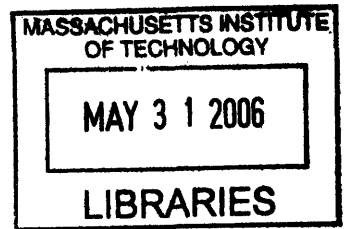
Submitted to the  
Engineering Systems Division and the  
Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements  
for the Degrees of

Master of Science in Technology and Policy and  
Master of Science in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology  
June 2006



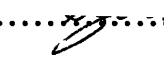
©2006 Massachusetts Institute of Technology. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

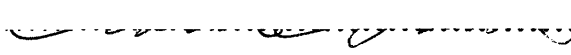
Signature of Author .....

May 11th, 2006  
Engineering Systems Division and  
Department of Electrical Engineering & Computer Science

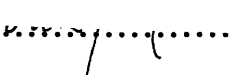
Certified by .....

  
Peter Szolovits  
Professor of Electrical Engineering & Computer Science and  
Professor of Health Sciences & Technology

Accepted by .....

  
Arthur C. Smith  
Chairman, EECS Department Committee on Graduate Theses

Accepted by .....

  
Dava J. Newman  
Professor of Aeronautics and Astronomics and Engineering Systems  
Director, Technology and Policy Program

**A Theory and Toolkit for the Mathematics of Privacy:  
Methods for Anonymizing Data while Minimizing Information Loss**

by  
Hooman Katirai

Submitted to the Engineering Systems Division  
and the Department of Electrical Engineering and Computer Science  
on May 11, 2006 in Partial Fulfillment of the Requirements for the Degrees of

Master of Science in Technology and Policy and  
Master of Science in Electrical Engineering and Computer Science

**ABSTRACT**

Privacy laws are an important facet of our society. But they can also serve as formidable barriers to medical research. The same laws that prevent casual disclosure of medical data have also made it difficult for researchers to access the information they need to conduct research into the causes of disease.

But it is possible to overcome some of these legal barriers through technology. The US law known as HIPAA, for example, allows medical records to be released to researchers without patient consent if the records are provably anonymized prior to their disclosure.

It is not enough for records to be seemingly anonymous. For example, one researcher estimates that 87.1% of the US population can be uniquely identified by the combination of their zip, gender, and date of birth – fields that most people would consider anonymous.

One promising technique for provably anonymizing records is called k-anonymity. It modifies each record so that it matches k other individuals in a population – where k is an arbitrary parameter. This is achieved by, for example, changing specific information such as a date of birth, to a less specific counterpart such as a year of birth. Previous studies have shown that achieving k-anonymity while minimizing information loss is an NP-hard problem; thus a brute force search is out of the question for most real world data sets.

In this thesis, we present an open source Java toolkit that seeks to anonymize data while minimizing information loss. It uses an optimization framework and methods typically used to attack NP-hard problems including greedy search and clustering strategies.

To test the toolkit a number of previously unpublished algorithms and information loss metrics have been implemented. These algorithms and measures are then empirically evaluated using a data set consisting of 1000 real patient medical records taken from a local hospital.

The theoretical contributions of this work include:

- (1) A new threat model for privacy – that allows an adversary’s capabilities to be modeled using a formalism called a virtual attack database.
- (2) Rationally defensible information loss measures – we show that previously published information loss measures are difficult to defend because they fall prey to what is known as the “weighted indexing problem.” To remedy this problem we propose a number of information-loss measures that are in principle more attractive than previously published measures.
- (3) Shown that suppression and generalization – two concepts that were previously thought to be distinct – are in fact the same thing; insofar as each generalization can be represented by a suppression and vice versa.
- (4) We show that Domain Generalization Hierarchies can be harvested to assist the construction of a Bayesian network to measure information loss.
- (5) A database can be thought of as a sub-sample of a population. We outline a technique that allows one to predict k-anonymity in a population. This allows us, under some conditions, to release records that match fewer than k individuals in a database while still achieving k-anonymity against an adversary according to some probability and confidence interval.

While we have chosen to focus our thesis on the anonymization of medical records, our methodologies, toolkit and command line tools are equally applicable to any tabular data such as the data one finds in relational databases – the most common type of database today.

Thesis Supervisor: Dr. Peter Szolovits

Title: Director, MIT Clinical Decision Making Group

**To Taymour, Mitra, Shadi and Justin**

## Table of Contents

Foreword .....	7
Acknowledgements.....	9
Motivation .....	11
The Proliferation of Anonymization Systems in the Medicine.....	11
Statement of Claims .....	13
Organization of this dissertation.....	14
A threat model for privacy.....	14
Methods of Anonymizing fields.....	17
Method 1: Ranging .....	17
Method 2: Binning.....	17
Method 3: Generalization .....	17
Related Work .....	23
Attacks on k-anonymity.....	24
Unsorted Matching Attack (Sweeney 2002).....	24
Linking Attack .....	24
Complementary Release Attack.....	25
Temporal attack .....	25
Attributes Occurring Less Than $k$ Times Attack .....	25
Measure Functions .....	26
A method for utilizing a Bayesian model as a measure function .....	31
The DSG Privacy Toolkit .....	34
Configuration Files .....	35
Posets.....	35
Columns.XML File.....	46
Ambiguators.....	47
Synthetic Data Generation.....	51
Command Line Interface .....	51
Experiments.....	52
Context .....	52

The Data Set.....	54
Majority Rule Voting.....	73
Condorcet Voting.....	73
Experimental Conclusions:.....	77
Conclusion.....	78
Future Directions / Questions .....	78
Appendix 1: .....	80
References .....	85

## **Foreword**

Who would have thought that a Computer Science thesis could be inspired by a lecture on public policy? That lecture was given by Frank Field, III in a class called “Introduction to Technology & Policy.” During that lecture he assigned us the task of devising a system for allocating kidneys to patients. That is, he handed us a stack of simulated medical records and asked us to write a memo outlining a system that would determine who should get the kidney first.

Knowing full well that the class was largely composed of engineers, Frank knew we would instinctively appeal to quantitative measures of need. Students in the class created mathematical expressions that would take into account a person’s age, the time they had been on a waiting list, and so forth, to determine who was in line to get the kidney first.

One team stood in front of the class talking about how their system was “highly objective” on the basis that anyone could enter the same numbers into their equation and get the same ranking.

Frank correctly pointed out that behind all these so-called “objective” systems” lay the highly subjective weights that we had assigned to different criteria such as a subject’s age. More generally, he said there is no objective way of summing heterogeneous quantities or criteria into a single score. In fact, he showed us examples where the ranking completely reversed itself – depending on which of several plausible methods were used for turning the criteria into scores. He therefore admonished us to stop using words like “objective” to describe our rankings and to instead admit what was taking place was a political process.

I carried this lesson with me while doing research for Staal Vinterbo at Harvard Medical School. Staal had recently devised an elegant formal framework for anonymizing information. His framework sought to anonymize information while maximize the value of the information for machine learning and statistical purposes.

Staal commissioned me and Robert Fischer to write a toolkit that implemented his ideas. And it is in the course of writing this toolkit that I developed the experiments laid out in this dissertation. I continued working on the problem in the following year when I joined Peter Szolovits’ Clinical Decision Making Group at the MIT Computer Science and Artificial Intelligence lab. It was Peter’s former student, Latanya Sweeney, whose seminal

thesis had created the entire field of what we now know as Computational Disclosure Control.

While implementing the toolkit I discovered that a scoring system Staal had published was based on weights – making it subject to the same weighted indexing problems pointed out earlier in that Public Policy lecture by Frank Field. I later examined other measures in the literature including Latanya Sweeney’s PREC metric and found it too was a weighted index because it could be interpreted as giving a weight of 1.0 to every column. It in effect assigns an equal score (or weight) to knowing if someone had HIV for example, or knowing if they were male or female.

These observations inspired a search for a more rationally defensible basis for valuing information loss in the anonymization process. The result may be described as a theory and toolkit described in this thesis.



## **Acknowledgements**

I am deeply grateful to the many people and organizations who made this thesis possible.

First and foremost, I'd like to thank Peter Szolovits, from the MIT Clinical Decision Making Group and my principal advisor. There is nothing more I could have asked for from an advisor. In just about every endeavor I did – I sought his wise counsel and advice. I am deeply grateful for his encyclopedic knowledge, for his sage advice and for introductions into his vast contact network. Peter greatly enriched my experience, and for this I will be forever grateful.

I would also like to thank Dr. Staal Vinterbo, from the Decision Systems Group at Brigham & Women's Hospital and Harvard Medical School, for funding me and advising me during my first year, teaching me new ideas and for inventing the theory that inspired this thesis. Staal has throughout my tenure at MIT offered many helpful suggestions and helpful criticism.

I am indebted to Dr. Megan Dirks, from Beth-Israel Deaconess Medical Center without whom access to real patient data would not have been possible. She single-handedly navigated the Institutional Review Process that resulted in the release of the data and helped me to understand the data's nuances.

I am also grateful to Isaac Kohane, MD-PhD, and Ken Mandl, MD-MPH from the Children's Hospital Informatics Program at Harvard Medical School for funding me for a summer internship to work on one of their projects called PING – a project that seeks to make multi-institution medical records a reality in a number of countries.

I would like to thank my colleague Stanley Trepetin, PhD from the MIT Clinical Decision Making group for valuable conversations and advice, and for help using DxCG (a software package used by healthcare companies to predict future costs).

I would also like to acknowledge the contribution of Dr. Robert Fischer, formerly from Harvard's Decision Systems Group at Brigham & Women's Hospital who wrote approximately 3400 of the 7200 lines of source code in the project. Robert authored the Partition algorithm, and wrote most of the code for the MatrixLattice, DefaultLattice, and RangeLattice classes. He also wrote algorithms to parse US Census data to build priors for

synthetic data generation. I am deeply grateful to Robert for his mentorship and for numerous helpful conversations. My contributions constitute the remainder of the library from 2003 to 2006 including the Greedy algorithm, the Command Line Tool, and all other Posets.

I would also like to thank the admission committee at the Technology & Policy program and the Department of Electrical Engineering and Computer Science for admitting me, and Stephanie Perrin (author of Canada's privacy law), Russell Samuels (my former Director at Zero-Knowledge), and Dr. Kostas Kontogiannis of the University of Waterloo for writing the letters of recommendation that helped get me here.

I am also grateful to my friends who greatly enriched my graduate school experience including: Emanuel Abbe, Vladimir Bychcovsky, Safa Sadeghpour, Carine Simon, Oleg Shamovsky, Armando Valdez and the members of The MIT Baha'i Association (Benjamin and Alison Dahl, David Gray, Rahmat Cholas, Ravi P, Zeba Wunderlich, and Dorri Ziai).

I also would like to also acknowledge DxCG Inc. who provided me and Stanley Trepetin a discounted copy of RiskSmart – an industrial grade software used by health insurance companies to predict patient costs for the purposes of disease management.

Lastly I would like to express my deep gratitude for having had all my expenses at MIT paid for by professors who ultimately received funds from the National Institutes of Health (NIH). In particular I would like thank Peter Szolovits who funded me during most of my tenure at MIT under NLM contract number N01-LM-3-3515, and Staal Vinterbo from Harvard Medical School for funding me during my first year under NLM contract R01-LM07273.

## **Motivation**

Privacy laws are an important facet of our society. But the same laws that prevent the casual disclosure of medical records have also made it increasingly difficult for researchers to obtain the information that they need to conduct research into the causes of disease. In this thesis, we present a theory and method for anonymizing information with a focus on the medical domain.

While we have chosen to focus our thesis on the anonymization of medical records the same techniques could potentially be used in any sector that seeks to anonymize tabular data (the type of data one finds in relational databases – the most common type of database today).

## **The Proliferation of Anonymization Systems in the Medicine**

Data privacy systems are quickly becoming an integral part of a broad range of medical systems. This is true for a number of reasons.

First, the disclosure of health information is strictly regulated in many jurisdictions and institutions are often legally required to apply privacy-enhancing transformations to health data prior to their disclosure to researchers. In the United States, for example, the Health Insurance Portability and Accountability Act (HIPAA) requires data to undergo either one of two privacy-enhancing processes prior to its disclosure.

In the first process, which may be termed *de-identification*, certain pre-specified fields such as name, address and social security number are removed. Although this process is sufficient to satisfy legal standards (United States Office of Health And Human Services 2003), the output of this process may still contain information that can be used to uniquely identify a member of the population. For instance, in one study, Sweeney estimated that 87.1% of the US population can be uniquely identified by the combination of their 5-digit zip code, gender, and date of birth (Sweeney 2002) because such records can be linked to publicly available databases such as voter lists, and driving records. To prove her point, Sweeney re-identified a series of supposedly anonymous medical records including one belonging to William Weld – the governor of Massachusetts at the time – using a voter list she purchased from the city of Cambridge, Massachusetts for a mere \$20 (Sweeney 2002).

The ease with which she obtained the public records she needed to re-identify his

record bears eloquent testimony of the inadequacy of de-identification techniques for preserving privacy. This motivates our discussion of the second privacy-preserving process acceptable under the HIPAA: anonymization.

An *anonymization* process renders a record “not individually identifiable” i.e. the record’s information cannot be used by an adversary “alone or in combination” with other “reasonably available information” to uniquely identify an individual (United States Office of Health And Human Services 2003). This thesis is focused on this second process: anonymization.

Unlike the first process, which could output data matching a single person in a relevant population, an anonymization process outputs records that match at least  $k$  individuals in a database of  $N$  records, where  $1 \leq k \leq N$ . Because  $k$  is an arbitrary parameter, it can be increased or decreased according to the sensitivity of the information and the needs of the application. For example, for an online advertising firm a low value of  $k$  may suffice, whereas in the healthcare domain, a higher value may be required.

The second reason for the proliferation of anonymization systems is that institutions are often hesitant, if not unwilling, to disclose private health information to third parties owing to legal liability and the possibility of negative publicity. Although, institutions will typically require third parties to sign confidentiality agreements, such assurances cannot defend against hacker attacks (Chin 2001), accidental disclosure (Walls 2000), or theft (Hines 2006) from insiders or outsiders. And because breaches in confidentiality can impede the original data provider’s ability to collect the data in the first place (John Hagel and Singer 1999; Mandl, Szolovits et al. 2001), while also exposing the original data provider to legal liability (Hodge, Gostin et al. 1999), institutions have strong incentives to mitigate such risks by anonymizing data prior to its disclosure.

Another motivation for pursuing anonymization systems is the long term vision of providing medical researchers, public health officials and policy makers with unfettered access to medical information without violating patient privacy (Gostin 1997). In many jurisdictions including the United States, researchers must justify their use of data to a review board prior to getting access to medical data – in what amounts to a slow and cumbersome process.

Research in automated record anonymization could speed the pace of medical research. For example, every hospital could have a record server that could offer anonymized records to researchers *on-demand* thus eliminating the scarcity of medical data.

Moreover, such anonymization systems could allow researchers to more freely engage in speculative and exploratory studies. Whereas current practice requires researchers to justify every information disclosure to a review board – a process that encourages researchers to limit requests to only their most promising studies, review boards may feel more comfortable releasing data for more speculative and exploratory studies with the advent of anonymization systems. In short, anonymization algorithms may provide the key to unlocking stores of medical information.

### **Statement of Claims**

In this dissertation I have introduced a number of new idea and technologies; in particular, I have:

1. Presented a new threat model for privacy and a concept called a “Virtual Attack Database” which can be used to formally model certain privacy threats.
2. Outlined a toolkit that can anonymize data and can measure the performance of various approaches to anonymizing data.
3. Introduced new anonymization algorithms and new measures of anonymization performance (to guide such algorithms) that are in principle more attractive than previously published measures.
4. Empirically measured the performance of various anonymization measures and algorithms on 1000 real hospital patient records.
5. Proposed a Bayesian network that can estimate (with a high degree of confidence), the  $k$ -anonymity of a record in a larger population. This enables researchers to safely release records that match less than  $k$  records in a database while still

assuring k-anonymity in a larger population. In sum, this procedure allows researchers to preserve more data.

6. Shown that generalization and suppression; which heretofore, have been thought to be distinct concepts are in fact the same thing insofar as every generalization can be represented as a suppression and vice versa. This was made possible by introducing a concept called an “augmented table” which includes all the fields implied by an original table.

### **Organization of this dissertation**

First, we will introduce a threat model for privacy. Once the threat model has been defined, we examine various methods of defending against these threats, including the technique of k-anonymity. Next, we examine how these methods fare under different scenarios, leading to a formal definition of the problem we wish to solve. We then introduce the Vinterbo framework for privacy, which attempts to anonymize rows in a table while maximizing the information value of each row according to some measure function for a desired level of k-anonymity. We then examine several measure functions present in the literature and discover that current measure functions are difficult to defend rationally, leading us to propose several new measure functions. Next we introduce several new algorithms for achieving k-anonymity. Our discussion then moves to a toolkit created by the author and Robert Fischer that implements our algorithms and measure functions. The performance of various algorithms and measures is then empirically evaluated according to a number of measures (both new and old) over both real and synthetic patient data. We then enter a discussion of how some of our initial assumptions can be relaxed to preserve more data.

### **A threat model for privacy**

We now define a threat model for privacy which we will refer to later in this paper. Before defining the problem formally, we will begin with a less formal definition. We begin by defining *privacy risk*.

The “privacy risk” of a piece of data (or datum) could be described as its “risk of re-identification.” It represents the risk of identifying an individual from a piece of

information given all other information available to an adversary (United States Office of Health And Human Services 2003).

For the purposes of developing a threat model we distinguish between *electronic privacy* and *non-electronic privacy*. To explain the difference, we concoct an example. Suppose you knew I liked to wear tall green hats (a rare trait). You could use this information to identify me in a public square. But unless this information (alone or in combination with other information available to my adversary) can be linked to some identifier in an electronic database, you cannot use this information to identify me using techniques such as “record linkage” (Felligi and Sunter 1969). This leads to a problem definition.

Suppose there exists a hypothetical database consisting of the amalgamation of all possible databases available to an adversary. Suppose this database also includes all fields that could be inferred by our adversary. We refer to this database as the *virtual attack database*. This virtual attack database is virtual in the sense that it may not actually be assembled but it could be constructed, if one or more databases were linked together using the family of record linkage techniques first described by Felligi and Sunter (Felligi and Sunter 1969).

This leads us to the notion of *privacy risk*, which could also be referred to as the risk of re-identification. If a field’s value does not exist in an adversary’s virtual attack database, then consequently there can be no *electronic privacy risk* associated with that data item for that given adversary. It is still possible, however, that another adversary will have a virtual attack database that could be linked to my field. But if no database fields in any virtual attack database could be linked to or inferred from my penchant for wearing tall green hats, then this knowledge about me does not carry any electronic privacy risk whatsoever.

Nonetheless, the knowledge that a person likes to wear tall green hats in a certain city still poses a risk insofar as such information could potentially be used to identify that person in a public square. We term this risk, a *general privacy risk* – as it requires no database to exist (or potentially exist) for a risk to be present. Put another way, the General Privacy of a datum encompasses both its electronic and non-electronic privacy risk.

In this thesis, we are solely focusing on reducing electronic privacy risk. For many applications, a consideration of general privacy reduces to an exercise in minimizing the electronic privacy risk. In other applications – particularly where an adversary can use a piece of information without the need to correlate it with data in a database – a general privacy risk exists in absence of an electronic privacy risk.

Having defined the problem informally, we now define the problem of electronic privacy risk formally. Let  $U$  represent the universal set from which individuals in a database are drawn. This universe will vary according to the application. For instance, if the database belongs to a small local clinic, the universe might consist of only the individuals living in the surrounding geographic areas served by the clinic, whereas if the database belongs to a mid-sized hospital, the universe may include individuals in a larger geographic area.

Let  $S \subseteq U$  represent the subset of individuals from the universe with records in a database,  $D$ . And let  $f(s)$  return a vector representing the fields in the database for record  $s$ . For simplicity, we overload notation to write  $f_i$  to represent the vector representing the  $i$ th individual in  $S$  where  $1 \leq |S|$  (assuming  $S$  has at least one record).

Today, one can readily buy a number of public records such as voter lists, birth records, driving records, and credit reports. Therefore, let us suppose an adversary has access to a “virtual attack database” containing records associated with a set of individuals  $R \subseteq U$ . Let  $g(r)$  represent a function that returns a vector whose elements represent the fields for the individual  $r \in R$  in the attack database. For simplicity we write  $g_i$  to represent the record corresponding to the  $i$ th individual in  $R$ . Also let us suppose that the fields returned by  $g(\cdot)$  include a social security number or any other group of one or more fields that in combination could uniquely identify a member of the population. Such collections of identifying fields we will hereafter term *quasi-identifiers* (Dalenius 1986) because such fields can be used in combination to uniquely identify an individual. In order for a group of fields to qualify as a quasi-identifier, the collection of fields must exist in a database that either exists or could be potentially be constructed by an adversary. Otherwise the adversary will have no way of using these fields to uncover the respondent’s identity.

If there are some overlapping fields in  $f(s)$  and  $g(r)$ , then the adversary can attempt to use record linkage techniques (Felligi and Sunter 1969) to link the records corresponding



to people in  $S$  to the individuals in  $R$ . We wish to find a privacy enhancing data transformation  $\Delta_i(\cdot)$  for each record  $i \in S$  such that each transformed record will match at least  $k$  individuals in  $U$ . We, however, cannot be sure what data the adversary has; i.e. the function  $g(\cdot)$  is unknown and thus also unknown are the quasi-identifiers used by the adversary. What we can be sure of, however, is that if there are at least  $k$  elements in  $S$  for whom information in  $f(s)$  matches  $\Delta(s)$  then there are at least  $k$  individuals in  $U$  that match that same criteria. Consequently, if we only release information matching  $k$  individuals in  $S$ , the adversary can narrow down the search to at most  $k$  individuals in  $R$ . In general, if a given record matches  $k$  records in  $R$ , we say that the record has “ $k$ -anonymity” – a phrase first coined by Sweeney (Sweeney 2003). A record’s  $k$ -anonymity can generally be increased by increasing the granularity of fields within the record. This parameter  $k$ , may be thought of as a quantifiable measure of privacy. The higher the number,  $k$ , the greater anonymity of the records output of the anonymization process.

### **Methods of Anonymizing fields**

In order to motivate the row anonymization algorithms in our toolkit we will go over some methods of anonymizing individual fields:

#### **Method 1: Ranging**

Ranging achieves  $k$ -anonymity by subsuming two ranges into a new range. For example, the two age ranges  $[10,25]$  and  $[8,20]$  could be ranged into  $[8,25]$ .

#### **Method 2: Binning**

Binning refers to the process of assigning various inputs into bins according to some criteria. A simple example of this is the discretization of ages such as 10 and 15 into pre-specified mutually exclusive age ranges such as  $[10,12]$  and  $[13,15]$ .

#### **Method 3: Generalization**

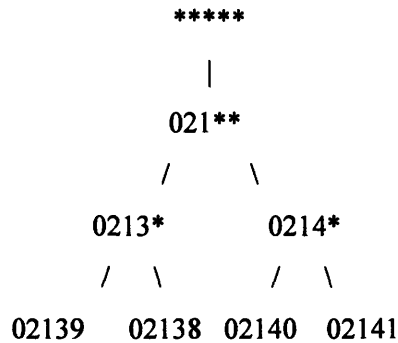
Generalization refers to the replacement of one data value, with a “more general, less specific value, that is faithful to the original” (Sweeney 2003). For example a city can be generalized to a less specific locale such as a county or a state; while a date of birth could be generalized to a year of birth or interval such as  $[25,30]$  representing an age range.

The generalizations that are possible for each attribute in a database S can be represented through a partially ordered set known as a Domain Generalization Hierarchy (DGH) (Sweeney 2003). Given an attribute  $A$  of a private table PT, Sweeney defines a **domain generalization hierarchy** (DGH) for an attribute  $A$  as a set of functions  $f_h : h=0, \dots, N-1$ :

$$\begin{array}{ccccccc}
 & f_0 & & f_1 & & & f_{N-1} \\
 A_0 & \longrightarrow & A_1 & \longrightarrow & \dots & \longrightarrow & A_N
 \end{array}$$

such that  $A=A_0$  and  $|A_N|=1$ . The latter requirement ensures that the final transformation generalizes to a single value – a useful trait to model deletion of the data, which in the statistical literature is referred to as “cell suppression.”

The DGH formalism models some hierarchies well but it fails to model others. Consider for example the hierarchy for zip code illustrated in Figure 1.



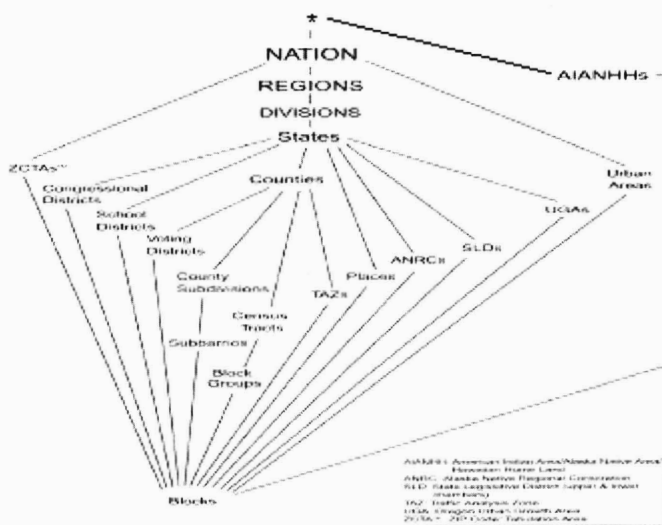
*Figure 1: An example of a domain generalization hierarchy, representing the possible generalizations of some US zip codes.*

This generalization hierarchy is perfectly captured by a DGH. On the other hand, there are some generalizations that a DGH cannot handle – particularly when a single value can generalize to more than one value. These situations are not merely theoretical curiosities; rather they do occur in real practice.

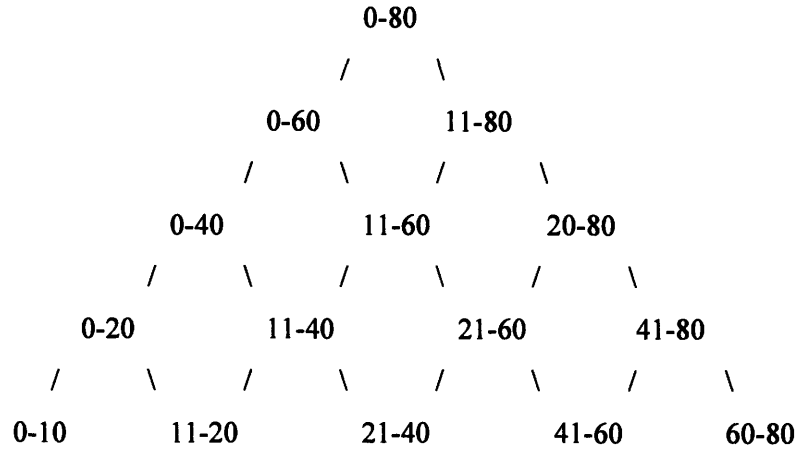
Consider for example, the generalization hierarchy corresponding to data distributed by the US Census Bureau (shown in Figure 2). In this example we see that a

census block can have multiple possible generalizations. In fact, Figure 2 illustrates that a census block can be generalized to no less than 13 different designations including a Zip Code or a Place (which encapsulates cities, boroughs and other geographic designations).

Another example demonstrating the need for multiple generalizations occurs when one wishes to discretize continuous values into a data structure called a lattice. The need for such a data structure arises, for example, when one wishes to bin ages into an age range – while still allowing those age ranges to be further generalizable to still wider age ranges. While one could impose a hierarchy where every single age range had only a single parent, the additional flexibility afforded by the lattice may capture more information.



**Figure 2: Generalization Hierarchy for the US Census Database.**



**Figure 3: An Age Lattice demonstrating that the ability for each node to have more than one parent is highly useful.**

Figure 3 illustrates how such a lattice could be constructed to handle a continuous variable such as age. It can be seen upon inspection that such a lattice can be constructed for any arbitrary discretization quanta.

The need for a lattice also becomes apparent when one wishes to use information theoretic measures such as mutual information as measures of information loss. Because such measures are not defined for continuous values, it becomes necessary to discretize the data. A lattice provides an elegant generalization hierarchy for discretizing such continuous values.

The latter three examples (age discretization, census generalization, and mutual information calculation) highlight the inadequacy of the DGH formalism to model multiple generalizations. To this end, we define a formalism called a Value Generalization Partially Ordered Set or VG-POSET. For simplicity we refer to this data structure as a Poset hereafter. A Poset is similar to a DGH, but it allows a single attribute to be generalized onto multiple possible attributes.

**Definition 1: Value Generalization Partially Ordered Set**

Formally, a Partially Ordered Value Generalization Hierarchy (VG-Poset) for an attribute A is defined as a set of functions  $f_{ij}: A_i \longrightarrow A_j$  such that  $N \geq i > j$  and for every  $i$  there exists at least one  $f_{ij}$ , and  $|A_N|=1$ . Again we use the latter requirement on  $A_N$  to force a single ultimate generalization representing the deletion of the cell.

**Definition 2: Measure function  $u(\cdot)$** 

A measure function  $u(\mathbf{r})$  for a record  $\mathbf{r}$  returns a real number representing the utility (or value) of the record  $\mathbf{r}$  to the user, where  $0 < u(\cdot) < \infty$ , with 0 representing the best possible score, and larger scores representing increasingly lower information value.

**Properties of a Measure Function**

We assume that one or many measure functions could be used. However, we wish to specify two minimum properties that a measure function should satisfy.

First, we require *monotonicity*; that is, if there exists a function  $f_{ij}$  in the domain generalization hierarchy (meaning that  $A_j$  is a more general form than  $A_i$  and that consequently  $A_j$  has less information content) and  $r_1 \in A_i$  and  $r_2 \in A_j$  then  $u(r_1) < u(r_2)$ .

Second, we require *transitivity*. Let  $X=(r_1, r_2, \dots, r_l)$  represent a vector of arbitrary length  $l$ . If each of the  $r_i$  represents an element in the  $\{A_{ij}\}$  such that for every pair  $(r_i, r_{i+1})$  there exists a function  $f_{i,i+1}$ , then the measure function must provide that for all  $u > v$ ,  $u(r_u) > u(r_v)$ . Put another way, if the vector  $X$  represents a path through the PODGH, then every generalization is required to have a “worse” score than its more specific predecessor; i.e. we are requiring strict monotonicity among these scores.

**Definition 3: Least Upper Bound (LUB)**

We define the Least Upper Bound of two values  $v_1$ , and  $v_2$ , i.e.  $LUB(v_1, v_2)$  for a Poset,  $D$  as follows. Let  $A = \{a_1, a_2, a_3, \dots, a_{|A|}\}$  be the set of values in  $D$  that are the ancestors of both  $v_1$  and  $v_2$ . For every element in  $A$ , let us take the measure of that value. Let  $M$  be the minimum measure value among these measures. Then the  $LUB$  of  $v_1$  and  $v_2$  is the subset of  $A$  whose measure values are equivalent to the lowest measure value; i.e.:

$$LUB(v_1, v_2) = \{ a_i \mid \iota(a_i) \leq \iota(a_j) \text{ for all } j \in \{1, 2, \dots, |A|\} \}$$

Similarly we define the LUB of two records  $X$  and  $Y$  as the pairwise LUB between the elements of  $X$  and  $Y$ ; i.e.:

$$\begin{aligned} LUB(X, Y) &= \{ LUB(\langle x_1, x_2, \dots, x_N \rangle, \langle y_1, y_2, \dots, y_N \rangle) \} \\ &= \langle LUB(x_1, y_1), LUB(x_2, y_2), \dots, LUB(x_N, y_N) \rangle \end{aligned}$$

Notice however that the LUB of two records is a set whose elements are also sets. This resultant, we term a “*LUB set*.”

**Definition 4: Least Upper Bound of a LUB Set and a Record R**

Let  $LS = \{\{X_{11}, X_{12}, \dots, X_{1j}\}, \{X_{21}, X_{22}, \dots, X_{2k}\}, \dots, \{X_{N1}, X_{N2}, \dots, X_{NL}\}\}$  be a LUBSet and let  $r = \langle X_1, X_2, \dots, X_N \rangle$  be a record. For simplicity of notation we re-define the LUBSet as  $\{Z_1, Z_2, \dots, Z_N\}$  where the  $Z_i$  are sets of rows, and let  $F = \{Z_1 \times \{X_1\}, Z_2 \times \{X_2\}, \dots, Z_N \times \{X_N\}\}$ . Again for convenience we re-name the sets  $F$  is composed of as  $F = \{F_1, F_2, \dots, F_N\}$ . We now define the LUB of the LUBSet  $LS$  and a record  $r$  as:

$$LUB_i(F_1 \times F_2 \times \dots \times F_N, R)$$

where  $LUB_i(X, y)$  represents the LUB of the  $i$ th element of set  $X$  with the vector representing a record  $y$  and  $i$  is chosen such that  $\iota(LUB_i(F_1 \times F_2 \times \dots \times F_N, R))$  is minimized.

We also define the LUB of more than two rows, written  $r_1, r_2, \dots, r_N$  as:

$$LUB(r_1, r_2, r_3, \dots, r_N) = LUB(\dots (LUB(LUB(r_1, r_2), r_3) \dots), r_N)$$

### **An optimization problem**

Let  $T$  represent a table where the columns represent attributes and the rows represent individuals in the population (i.e. patients, customers, et cetera). We wish to create a new table  $T'$  by replacing each row,  $r$ , in  $T$  with the LUB of the original row and  $k-1$  other rows, written  $r_1, r_2, \dots, r_{k-1}$  such that the measure,  $l(\cdot)$  for the  $LUB(r, r_1, r_2, \dots, r_{k-1})$  is minimized.

As shown in Vinterbo (Vinterbo 2002), achieving k-anonymity is an NP-hard problem. As a result, all practical algorithms must make numerous choices at various points in the ambiguity process. What attributes should be sacrificed (i.e. generalized) and which should be kept? Unless the algorithm has some notion of information value, it has no basis to guide such decisions.

### **Our Extensions to the Vinterbo Framework**

The Vinterbo framework provides an elegant way of modeling the k-anonymity problem. But a researcher using the Vinterbo framework is faced with the problem of defining  $l(\cdot)$ , the measure function. In this dissertation we explore different measure functions, in search of ones that are rationally defensible.

### **Related Work**

1. *DataFly* (Sweeney) – DataFly was the first published k-ambiguity algorithm. It has no notion of information value and is therefore a blind algorithm.
2. K-Similar (Sweeney) – K-similar also achieves k-anonymity. K-similar, unlike DataFly, does utilize a notion of information value; however, we show that the information measure used by the K-similar algorithm falls into a category known as a “weighted index” that is difficult to justify rationally; we later present measure functions which have a more sound rational basis.

3.  $\mu$ -argus and  $\tau$ -argus (Hunderpool, et al.) - Developed at Statistics Netherlands, this system was proven by Sweeney to not provide sufficient k-anonymity.

Our work differs from the others through (1) the use of a formal framework; (2) our exposition on why the measure functions of existing algorithms cannot be defended (3) the introduction of new ambiguation algorithms and defensible measure functions (4) the use of empirical measures of real data and (5) the theoretical contributions listed in our statement of claims (see page 13). Further the source code for this project has been released as an open source project, whereas the source code for the above projects are not available.

### **Attacks on k-anonymity**

Having defined how we intend to achieve k-anonymity, we now define attacks against our system.

### **Unsorted Matching Attack (Sweeney 2002)**

A table can be k-anonymized in many different ways. Thus it is possible to release many different anonymized versions of the same table. If the rows in those anonymized tables are listed in the same order; than one can combine the rows from different tables to infer more information that was otherwise possible by examining each individual row. To thwart this attack one can simply randomize the order in which the rows are released in each disclosure.

### **Linking Attack**

This Attack (Felligi and Sunter 1969) recognizes that records in different databases can be linked together. The attack begins by calibrating a probability model whose parameters include the probability of finding similarities and differences between two records referring to the same individual. The similarities might include factors such as “having the same last name” while the differences might include “having a spelling error with a string edit distance of 1.” Using this model one can estimate the probability that records in two different databases refer to the same individual. An arbitrary probability threshold is set above which two records are said to be a match. For example, one might declare that “all records with 95% likelihood of being a match will be considered a match.”



### **Complementary Release Attack**

This attack recognizes that if one discloses an anonymized version of a table then that table should be considered to be “joining external information” available to an adversary (Sweeney 2002). The implication is that if one releases some portion of a table in the present (which we shall call PT) that may limit one’s ability to safely release an anonymized version of the same table in the future (which we shall call FT). One solution to this problem is to base FT on PT; or to consider PT as a part of the quasi-identifier.

### **Temporal attack**

Because tables tend to change over time, subsequent releases of a table may allow one to draw inferences (Sweeney 2002). The solution is to not anonymize the current table but instead base the disclosure on the union of the previously disclosed tables and the new rows added to the table.

### **Attributes Occurring Less Than $k$ Times Attack**

Attributes occurring less than  $k$  times can possibly lead to re-identification (Sweeney 2002). Consider, for example, a table corresponding to the inhabitants of a village where all people have the race “white” whereas one person has the race “black.” In a  $k$ -anonymized version of the original table, the black person’s race will be the only one that is suppressed. Thus, we can readily re-identify this record, even if the race attribute is suppressed and regardless of how high a level of  $k$  was selected. One can defend from this attack by deleting all rows containing values occurring less than  $k$  times.

## Measure Functions

There are a number of possible measure functions of information value. In the context of a hierarchy, Sweeney proposed the PREC measure for a table as follows:

$$prec_{table}(RT) = 1 - \frac{\sum_{i=1}^{N_A} \sum_{j=1}^N \frac{h_{i,j}}{|DGH_{Ai}|}}{|PT| \bullet |N_A|} \quad (1)$$

Where:

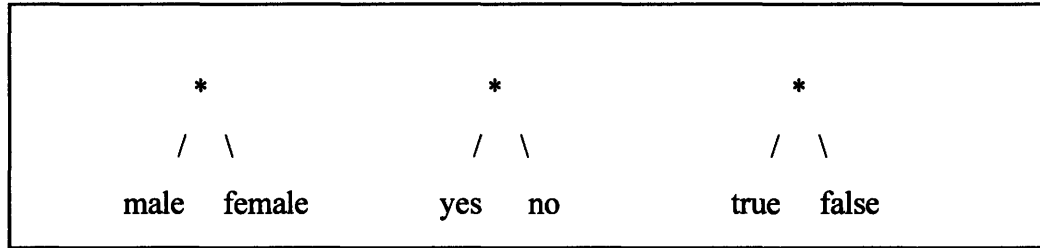
$R$	is a released row
$ PT $	is the number of rows in the original table,
$ N_A $	is the number of columns in the original table,
$ DGH_{Ai} $	is the height in the domain generalization hierarchy for attribute $i$ where the leaves are considered height 0.
$h_{i,j}$	is the height of the node in the Domain Generalization Hierarchy corresponding to the value of the cell $(i,j)$ in the generalized table.

To gain insight into the workings of the PREC measure, we introduce equation 2. Whereas equation 1 is a measure that applies to an entire table, the measure listed in equation 2 can be interpreted as the PREC measure for an individual row. Put another way, when averaged over all rows, Equation 2 reduces to the PREC measure listed in Equation (1).

$$prec_{row}(RT) = 1 - \frac{\sum_{i=1}^N \frac{h_i}{|DGH_{Ai}|}}{|N_A|} \quad (2)$$

Although the PREC measure would seem to be a reasonable measure of information value it falls into a category of measures called “weighted indexes” which are known to be rationally indefensible. We illustrate this by example. Suppose we have a database with three attributes; namely *has\_renal\_failure*, *has\_hiv* (a binary field indicating

the presence of the HIV virus) and gender. Assume all three fields are associated with a 2 level domain value hierarchy shown in Figure 4.



**Figure 4:** Domain Generalization Hierarchies for gender, has\_renal\_failure and has\_hiv

Now consider the rows of Figure 5. The first row of the table represents the original row, while the second, third and fourth rows respectively represent the original row where the fields of gender, HIV or renal failure have been generalized. The PREC measure does not distinguish between these three generalizations insofar as it assigns the same PREC score to each of these three rows. More pointedly, to the PREC measure the fields gender, HIV and renal failure are equally valuable.

But upon what basis does this algorithm assign the same value to preservation of the gender field as it assigns to the preservation of HIV? And if HIV shouldn't be equally valuable as gender, how many times more valuable *should* HIV be than gender? Whatever answers are given to these rhetorical questions are likely to lack a rational basis.

<b>Description</b>	<b>Gender</b>	<b>HIV</b>	<b>Renal Failure</b>	<b>PREC</b>
Original Row	M	False	True	1.00
Gender Cell Generalized	*	False	True	0.66
Gender Cell Generalized	<i>M</i>	*	<i>True</i>	0.66
<i>Gender Cell Generalized</i>	<i>M</i>	<i>False</i>	*	0.66

**Figure 5: The frailty of weighted indexing.** The first row represents the original row, whereas each subsequent row is a mirror of the original but with a different attribute suppressed. Although rows two through four suppress different attributes of the original row, their PREC measure is the same. This indicates that the PREC measure implicitly considers the columns Gender, HIV and Renal Failure to be of equal value.

The problem just outlined is sometimes referred to as the “weighted indexing problem” and it arises whenever quantities of non-convertible heterogeneous units are summed into a single unit. It fails to be resolved if heterogeneous units are multiplied by a weight before the summation takes place. A treatment of this problem is found in Field’s PhD thesis (Frank Remson Field 1985). Field shows that weighted indexing cannot be rationally defended as a measure of utility, and that neither normalization nor the conversion to ranks can solve the fundamental problem.

Another possible weighting method is presented as an example by Vinterbo (Vinterbo 2002). In this system the leaves of the Poset (or Domain Generalization Hierarchy) are assigned a value of 0 (which is considered a perfect score) while subsequent levels are assigned progressively higher values such that the monotonicity property of the measure function is satisfied. To compute the value of a row one sums the values of nodes corresponding to values in each column. This again constitutes a weighted index (with a weight of 1.0) for each column.

In many respects, this metric mirrors Sweeney’s PREC metric insofar as the value of a data item is totally dependent on its level in the hierarchy. In fact, if one sets the measure of each node in the hierarchy to its height in the hierarchy, one obtains the same result as that illustrated in Figure 3 – where the fields HIV, gender, and renal failure are implicitly considered by the measure function to be equally valuable.

If weighted indexes cannot be rationally defended are there any information measures that can be defended? We propose a number of measures in order of increasing defensibility.

**Proposed Measure 1: Mutual Information Of A Column With Respect To A Target Variable**

Since the purpose for disclosing the data in the first place is to build predictive models, we propose using the Mutual Information with respect to the predicted variable as a column weight. Of course, this assumes we know in advance what variable we wish to predict. In such a situation this would seem to be a more rational approach than simply using a weight of 1.0 for all columns, as was done in earlier examples. The mutual information between a variable X and a predicted class variable C is defined as:

$$MI(C; X) = \sum_i \sum_j p(X_j | C_i) p(C_i) \log_2 \left( \frac{p(X_j | C_i)}{p(X_j)} \right)$$

where X is a random vector representing the rows in the database, and C is a class variable we wish to predict. The above equation however is a metric for an entire column; whereas we need a measure for a specific row. We propose that the mutual information scores for each column be summed, in order to calculate this metric for a row.

For example, suppose we have a table where each row represents a patient and where the columns represent attributes of a patient. If one of the columns is a binary variable representing the diagnosis of cancer, then we can use the mutual information of all variables with respect to the binary variable cancer as our information measure.

The use of mutual information as an information value measure is not a panacea. It requires us to declare in advance one or more variables that will be predicted by the predictive models that the data will be used to build. But in many applications, we simple

may not know how the data will be used and so will be unable to determine what variable should be optimized.

Moreover, the use of the mutual information metric may have unexpected effects on future disclosures owing to the consequences of the Complementary Release Attack (Sweeney 2002) discussed on page 25. In the context of using mutual information, the implication of this attack takes on additional connotations: by releasing a table optimized for predictive modeling of one variable, we may also limit our ability to release information optimized for other variables

The mutual information metric also suffers from another serious problem. If the table we are trying to anonymized has more than one column (other than the column we are trying to predict) we are again faced with the weighted indexing problem when summing the measures from the individual columns into a measure for the entire row. It is perfectly defensible to sum the mutual information scores of the columns into a single score if the columns contain no redundant information (or put another way, if the columns were generated wholly independent of each other). But this assumption is almost certainly violated in practice. Nonetheless, the fact that there are situations where the columns can be defensibly summed, and the fact that the weights are generated from the data itself, makes this measure more defensible than totally arbitrary weight of 1.0.

### **Proposed Measure 2: The Degradation in Performance of a Predictive Model**

Suppose we know in advance the variable that the data will be used to predict. Suppose we also have a predictive model that can handle missing values (although this assumption will be later relaxed). If the output of the model for each row on the fully identifiable data is taken as a “gold standard” – then deviations from this output can be seen as error introduced by the anonymization process. This measure can be calculated at the individual row level (as the deviation from the gold standard) or at a table level (using measures such as the mean, average, total error and standard deviation). Predictive models particularly suited to this purpose include Bayesian networks and Decision trees since they can both handle missing information.

It is also possible to use predictive models that cannot normally handle missing input values if the missing data consist of categorical values (as opposed to continuous ones). Techniques for handling missing information include the Expectation Maximization (Dempster, Laird et al. 1977), Gibbs Sampling (Geman and Geman 1984), Multiple Imputation (J.L Schafer 1999) and Robust Bayes Estimation (Ramoni and Sebastiani 2001).

**A method for utilizing a Bayesian model as a measure function**

Here we propose a method for a constructing a Bayesian model that could serve as the predictive model for measure 2. It will also be shown that this model can also be used to predict missing values. The basic idea is to harvest the structural information inherent in a DGH or Poset to aid the discovery of the structure of our model.

Before getting into the details we first introduce the idea of *implied fields* in a table. The concept of *implied fields* is useful when modeling generalization using a Poset or DGH. Consider for example a simple table such as that found in Figure 6, and suppose that Race and HIV cannot be generalized (except by suppression) but that the field Zip can be generalized according to the DGH set out in Figure 1. Suppose however, that we augment this table with the “implied fields” zip4, and zip3 representing the 4-digit and 3-digit generalizations of the 5-digit zip codes in column zip5. We call this table, an *augmented table* because the original table has been augmented to include all implied fields.

<b>Race</b>	<b>HIV</b>	<b>Zip5</b>
White	Yes	02139
Black	No	02138
Asian	Yes	98052

**Figure 6: Original Table**

Race	HIV	Zip5	Zip4	Zip3
White	Yes		0123	021
Black	No	02138	0213	012
Asian	Yes	98052	9805	980

**Figure 7: An Augmented Table for a simple table containing a Zip code where the first row has been generalized to a 4 digit zip.**

Herein lies the elegance of this model. In this augmented table, every generalization is represented by exactly one cell suppression. In Figure 7 for example, a row had the value “02139” in the zip5 field. To generalize this field into its four-digit counterpart (i.e. “0213”) we simply suppress the zip5 field while leaving the values in the zip4 and zip3 intact. It can also be seen upon inspection that every suppression can be represented as a generalization in an augmented table.

Earlier, we mentioned that the information loss of an anonymization algorithm on a data set can be measured using a predictive model; however, we have not show how to build such a model. As will be shown, one can harvest the structure of a DGH (using an Augmented Table) to build a Bayesian Network which can serve as the predictive model.

The Bayesian Network can be constructed using the following process:

- 1. Construct a model using the fully identifiable information.**

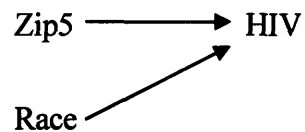
This model is constructed using techniques commonly used by Bayesian practitioners. One common practice is to split a data set into two parts (Mitchell 1997). The first (the “training set”) is used to discover the structure of the Bayesian network and to train the model. The second, (the “test set”) is used to test the model. The Bayes network is often constructed by beginning with no fields, and



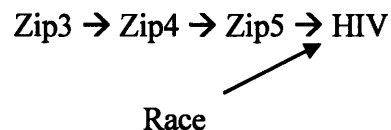
adding fields (one at a time) ordered by their mutual information scores with respect to the variable being predicted<sup>1</sup>. As each field is added to the Bayesian Network, the Bayesian network is retrained using the training set. When the performance of the model on the training set begins to decrease, no further fields are added and the performance of the network is then tested on the test set. If the model performs well on the test set, the model is said to be “generalizable” insofar as it performed well on a data set on which it was neither constructed nor trained.

## 2. Enhance the Bayesian Model using Implied Fields and Information from the Poset

Suppose in the last step a simple model was constructed with a node configuration as follows:



For every field in the model constructed at step 1, we add nodes in the Bayesian network corresponding to its parents in the DGH or Poset. For instance, in the latter example we would have:



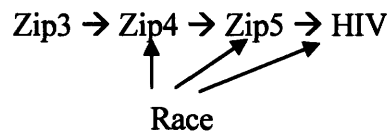
That is, 3-digit zip code would try to predict the 4-digit zip code (when the 4-digit zip code was missing), and the 4-digit zip code would try to predict the 5-digit zip code (when the 5-digit zip code was missing).

---

<sup>1</sup> A procedure mentioned by Marco Ramoni, a Bayesian specialist and Professor of Pediatrics and Medicine at Harvard Medical School.

### 3. Add Further fields to the model as appropriate

One may discover, for example, that the field “race” may help predict one or more of zip3, zip4 or zip5. Thus, it is appropriate to have a third step where one adds nodes to the Bayes network, using the Mutual Information procedure outlined in step 1. The final outcome of this process may look like something like this:



The link from race to Zip5 and Zip4 would seem to indicate that race can predict one’s zip codes with a certain specificity – an unsurprising conclusion when one considers that many neighborhoods contain a preponderance of people from a given race.

The above procedure is but one way of discovering the structure of a Bayesian network; however, we believe this procedure is advantageous insofar as it harvests the structure of the DGH to aid the discovery process. Given that there is, at present, no generally accepted method for discovering the structure of a Bayesian network, it would seem that this procedure offers a good start.

It would also seem that such a network could also be used to attack k-anonymity, however, such application is beyond the scope of this dissertation.

#### The DSG Privacy Toolkit

The DSG Privacy Toolkit is a Java API and a collection of command line tools written by the author and Robert Fischer that anonymizes information using an extended version of the Vinterbo framework. We now outline the toolkit and some of the design choices made.

As noted earlier, the DSG Privacy Toolkit achieves *k*-ambiguity via generalization. In order to ambiguate data using the toolkit 3 things need to be specified: (1) the data to be

anonymized (2) partially-ordered generalization hierarchies for each attribute, and (3) an ambiguity algorithm. We now outline each of these three aspects in greater detail.

### ***Value Generalization Partially Ordered Set (VG-Posets or Posets for short)***

Generalization is implemented via the Least Upper Bound (LUB) operator (as earlier defined). In order to specify the Least Upper Bound operator for an attribute a partially ordered set (Poset) is necessary. Our toolkit has several built-in Posets, all of which implement the Poset interface.

Different fields can be better represented by different kinds of Partially Ordered Sets. In the DSG toolkit, these Posets are represented by classes implementing the Poset interface. Moreover, each type of Poset has associated with it a particular type of node that is used to represent values in the hierarchy. These nodes inherit from the PosetNode class.

### **Configuration Files**

The DSG toolkit's command line interface uses two configuration files – Column.xml and Hierarchy.xml. Both are required to load data from a Comma Separated Values (CSV) file. The Hierarchy.xml file specifies VG-Posets and Domain Generalization Hierarchies; whereas the Columns.xml file binds these aforementioned Posets to specific columns in a CSV file. In the examples that follow we describe all built-in Posets supported by the toolkit and how the configuration files can be configured to instantiate each possible Poset.

### **Posets**

Here we present different classes associated with different types of Posets. For each Poset we present an overview of how it works and the situations in which it is useful. The toolkit can be used in two different ways: as an API and as a command line tool. As a result we provide two ways of instantiating each class (1) via source code (which is useful when the toolkit is being used via an API) and (2) via the Hierarchy.xml configuration file, when the toolkit is being used as a standalone anonymization tool.

### *RangePoset and RangeNode*

A RangePoset is useful for describing attributes that represent ranges. In particular it implements ranges that satisfy the property that the LUB of [A,B] and [C,D] is [min(A,C), max(B,D)] where A,B,C, and D are double precision floating point numbers that were encapsulated into a RangeNode class. A RangePoset is particularly suited for describing a hierarchy of age ranges because the upper bound of two age ranges such as [10,20] and [15,30] satisfies the latter property (i.e. the upper bound would be [10,30]). We do not call the latter, however, a *least* upper bound, because age ranges do not always satisfy the above LUB property. Consider, for example, the non-overlapping age ranges [10,12] and [14,15]; here the LUB is not [10,15] since the element 13 is not included in the *least* upper bound. Nonetheless [10,15] can be considered an upper bound, and so a RangePoset can be used in that manner. RangePoset can be constructed as follows: RangePoset(double low, double high) where low and high respectively represent the lower and upper limits of allowable values. The LUB function of the RangePoset class accepts RangeNode objects or a LUBset class.

```
// create a new RangePoset that can hold
// people within the ages of 10 to 90
RangePoset AgeRangePoset = new RangePoset(10,90);

// Create a node representing an
// age range from 10 to 30 years of age
RangeNode AgeRange1 = new RangeNode(10,30);

// Create a node representing an
// age range from 20 to 34 years of age
RangeNode AgeRange2 = new RangeNode(20,35);

// find the LUB of the two age ranges
RangeNode result = AgeRange1.lub(AgeRange2);
```

#### **Example 1: How a RangePoset and RangeNode are used**

The following is an example of how one might create a RangePoset in the Hierarchy.xml file:

```
<RangePoset Name="Age_range_poset" LowerBound="0" UpperBound="120" MaxDiff="25" />
```

#### **Example 2: How a RangePoset can be instantiated in the Hierarchy.xml file**

In the above example we have created a RangePoset called “Age\_range\_poset” whose ranges can span from 0 to 120 and where the maximum difference in ages before the RangeNode assumes an information value of 0 is 25.

#### *MatrixPoset and MultiNode*

Partially ordered sets that can take on a finite number of values (such as the one shown in Figure 3) can be thought of as a directed graph represented by a collection of nodes and directed edges. The nodes represent items in the hierarchy, and the edges represent less than or equal to relationships within the hierarchy. The MatrixPoset allows one to specify such Posets. It uses the MatrixNode class to specify specific nodes in the Poset. To speed computations, the less than or equal to relationship (LEQ) is pre-computed for all possible pairs of values in the MatrixLattice. To more rapidly calculate this table, we note that the lookup table listing the LEQ relationship is the transitive closure of the adjacency matrix between all nodes – a realization that enables us to take advantage of relatively efficient algorithms for transitive closure in the literature. While such pre-computation increases the initial start-up time for our anonymization process, we have found that pre-computation resulted in considerable performance improvements – changing our run times from hours to minutes.

```

// create a MatrixPoset to represent a hierarchy
// of ZIP codes

// create a new MatrixPoset that can hold 8 nodes
MatrixPoset MP = new MatrixPoset(8);

// add the nodes
MP.addNode("",4);
MP.addNode("021",3);
MP.addNode("0213",2);
MP.addNode("0214",2);
MP.addNode("02139",1);
MP.addNode("02138",1);
MP.addNode("02140",1);
MP.addNode("02141",1);

// set the less than or equal to relationships
MP.setLeq(MP.getNode(""), MP.getNode("021"));
MP.setLeq(MP.getNode("021"), MP.getNode("0213"));
MP.setLeq(MP.getNode("021"), MP.getNode("0214"));
MP.setLeq(MP.getNode("0213"),MP.getNode("02138"));
MP.setLeq(MP.getNode("0213"),MP.getNode("02139"));
MP.setLeq(MP.getNode("0214"),MP.getNode("02140"));
MP.setLeq(MP.getNode("0214"),MP.getNode("02141"));

// pre-calculate LUB values
MP.setTclosure();

// find the LUB of 0213 and 0214
MatrixNode MN1 = new MatrixNode(MP.getNode("0213"))
MatrixNode MN2= new MatrixNode(MP.getNode("0214"))
LUBSet result = MP.getLubset (MN1,MN2);

// result now holds a LubSet containing the node "021"

```

**Example 3: How a MatrixPoset is used**

The following is an example of how one might create a MatrixPoset in the Hierarchy.xml file:

```
<MatrixPoset Name="ICD9_Codes" RootNode="ID9000000">
  <Node ID="ID9000000" Name="*" Value="1" Parents="ID9000000"/>
  <Node ID="ID9000001" Name="008" Value="1" Parents="ID9000000"/>
  <Node ID="ID9000002" Name="031" Value="1" Parents="ID9000000"/>
  <Node ID="ID9000003" Name="038" Value="1" Parents="ID9000000"/>
</MatrixPoset>
```

**Example 4: How a MatrixPoset can be Instantiated in the Hierarchy.xml file**

In the latter example, we have a MatrixPoset of 3 Diagnostic Related Grouping (DRG) codes representing different medical procedures. The root node in the hierarchy represents the suppression of the cell. As always, this root node is its own parent and we've given it the name "\*" (a symbol often used to denote a wildcard) to denote that a deleted cell could match any value. The remaining three nodes represent the DRG codes allowable in the data set – all of whom list the root node as their parent.

#### *PowerPoset*

A group of one or more binary fields can be represented by a PowerPoset. The PowerPoset assumes that the LUB of two bit-vectors  $a$  and  $b$  is  $a \wedge b$ , where the  $\wedge$  operator represents a bitwise AND operator. As an example, let us suppose that  $a$  and  $b$  are both two-bit bit vectors, where the first bit represents the presence of "HIV" and the second represents the presence of renal failure. The LUB of  $a$  and  $b$  will be the traits common to both vectors  $a$  and  $b$ . In other words, the LUB of  $a$  and  $b$  is analogous to a set intersection operator.

```

// create a new PowerSet that can hold
// 3 boolean fields
PowerSet PS = new PowerSet(3);

// Create 2 power nodes of length 3
// (initialized to binary '000')
PowerNode PN1 = new PowerNode(3);
PowerNode PN2 = new PowerNode(3);

// set the second and fifth positions of
// the two Boolean vectors to '1'.
PN1.set(2,1); // turns on the second bit of PN1
PN1.set(3,1); // turns on the third bit of PN1
PN2.set(3,1); // turns on the third bit of PN2

PowerNode result = PS.lub(PN1,PN2);

// result now holds a PowerNode with binary value 011

```

**Example 5: How a PowerPoset is used**

The following is an example of how one might create a PowerPoset in the Hierarchy.xml file:

```
<PowerPoset Name="Has_HIV" Bits="1"/>
```

**Example 6: How a PowerPoset can be instantiated in the Hierarchy.xml file**

In the latter example, we initiated a Poset for a single Boolean bit that will store whether or not a person has HIV.

*DateOfBirthPoset*

A DateOfBirthPoset is useful in situations where one wants to preserve as much information about a date of birth as possible. Its LUB operator preserves as much information as is common to its two inputs. If the two dates of birth do not occur in the same year, it converts both inputs into age ranges to see if they fall into the same range, in which case it returns the date of birth of a person who was born at the midpoint of that age range. Finally, if no commonality can be found at the age range level, it suppresses the cell. The *DateOfBirthNode* keeps track of the granularity of the node (i.e. day of birth, month of birth, year of birth, age range, or suppression). The purpose of this to



facilitate calculating LUBSets. A LUBSet of several different dates of birth falling into the same age range will not change if an additional DateOfBirthNode falling within the same age range is added.

Here we give a formal definition for the LUB operator. Let a date be composed of the triplet  $(d,m,y)$  where:

$d$  is the day  
 $m$  is the month  
 $y$  is the year

And let  $a_1, a_2, \dots, a_N$  represent a series of ordered pairs,  $(a_{11}, a_{12}), \dots, (a_{N1}, a_{N2})$ , representing age ranges  $[a_{11}, a_{12}], \dots, [a_{N1}, a_{N2}]$  such that  $a_{i1} = a_{(i-1)2} + 1$ .

And let  $D_B = (d_B, m_B, y_B)$  represent a base date against which other dates will be compared

Then LUB of dates of birth  $D_1 = (d_1, m_1, y_1), D_2 = (d_2, m_2, y_2), \dots, D_N = (d_N, m_N, y_N)$  is

$(d_1, m_1, y_1)$  if  $D_1 = D_2 = \dots = D_N$   
 $(15, m_1, y_1)$  if the days are different but the months and years are the same  
 $(15, 6, y_1)$  if the months are different but the years are the same  
 $(q, r, s)$  if the years are different,  $D_1, D_2, \dots, D_N$  represent the age of someone who at  $D_b$  will be greater than  $a_{i1}$  years of age and less than  $a_{i2}$  years of age and  $q, r,$  and  $s$  respectively represent day, month and year of birth of a person who is  $(a_{i1} + a_{i2})/2$  years of age at time  $D_b$ .

```

// age ranges corresponding to [0,5], [6,10]
// [11,15],[16,20] ... etc ...
int [] rgAge = new int [] {5,10,15,20,25,30,35,40,45,50};

// creates Poset using today's date as the base date
DateOfBirthPoset DOBP = new DateOfBirthPoset
    (new GregorianCalendar(), rgAge);

// illustrates creating DateOfBirth nodes using
// different date formats
DateOfBirthNode A = new DateOfBirthNode("2/13/1978", rgAge);
DateOfBirthNode B = new DateOfBirthNode("19780213", rgAge);
DateOfBirthNode C = new DateOfBirthNode("19780201", rgAge);
DateOfBirthNode D = new DateOfBirthNode("19781113", rgAge);
DateOfBirthNode E = new DateOfBirthNode("19791013", rgAge);

DateOfBirthNode T1 = A.lub(B,rgAge); // result: 19780213
DateOfBirthNode T2 = C.lub(A,rgAge); // result: 19780215
DateOfBirthNode T4 = D.lub(E,rgAge); // result: date of birth
                                        // corresponding to
                                        // midpoint of age range
                                        // [25,30]

```

**Figure 8: How a DateOfBirthPoset is used**

The following is an example of how one might create a PowerPoset in the Hierarchy.xml file:

```
<DateOfBirthPoset Name="Age" BaseDate="20060101" AgeRanges="5,30,50,70,90"/>
```

**Example 7: How a PowerPoset can be instantiated in the Hierarchy.xml file**

In the above example we have create an DateOfBirthPoset with a base date of Jan 1<sup>st</sup>, 2006. The age ranges it uses are [0,5], [6,30], [31,50], [51,70], and [71,90].

### *SparseMatrixPoset*

A `SparseMatrix` is sparse in the sense that it compactly represents a matrix. The matrix it represents has all possible values of a `MatrixPoset` on one axis, and some quantity (such as frequency) on the other. A `SparseMatrixPoset` is useful when one needs to perform a join between two or more tables. Consider for example the case where a patient's demographic details reside in one table, and their diagnostic codes reside in another.

The `SparseMatrixPoset` can represent these diagnoses as a set of pairs  $(MN, f)$  where `MN` is a `MatrixNode` and `f` is the frequency that the contents of the node appears. For example, if a patient has 10 cardiac dysrhythmias (which are represented by the ICD9-CM code 427.89) then that would be represented by the pair:  $(427.89, 10)$ . A `SparseMatrixPoset` then can be used to hold the frequencies of occurrence of all diagnoses in the diagnosis table for each patient.

In order to create a `SparseMatrixPoset` one must specify the maximum number of  $(MN, f)$  pairs that can occur in a given row in addition to passing a reference to an instance of a `MatrixPoset` class (which in our latter example would define all possible ICD-9-CM codes).

In the `Hierarchy.XML` Configuration file one can create a `SparseMatrixPoset` as follows:

```
<SparseMatrixPoset Name="SparseMatrixPoset_ICD9"  
    MatrixLatticeName="ICD9_Codes" Columns="39"/>
```

Where:

<code>Name</code>	is the name given to this <code>SparseMatrixPoset</code>
<code>MatrixLatticeName</code>	is the matrix lattice used to define the LUB of the <code>MatrixNodes</code> stored in this <code>Poset</code> .
<code>Columns</code>	is the maximum number of Name-Value pairs that can be stored in a given row.

The following is an example of how one might create a SparseMatrixPoset in the Hierarchy.xml file:

```
<SparseMatrixPoset Name="SparseMatrixPoset_ICD9" MatrixLatticeName="ICD9_Codes" Columns="39"/>
```

**Example 8: How a PowerPoset can be instantiated in the Hierarchy.xml file**

In the above example, we have created a SparseMatrixPoset. The values it can assume are defined in the MatrixPoset named ICD9\_Codes. This SparseMatrixPoset contains 39 pairs of columns. Each pair stores an ICD9 code of a diagnosis together with the frequency with which that diagnoses occurred for that patient.

*PassThroughPoset*

A PassThroughPoset is useful when one wishes to add fields to a row that should not be anonymized. The LUB of a PassThroughNode and another node is the unchanged original node. The measure of a PassThroughPoset is always 0 because its information can never be degraded.

```
<PassThroughPoset Name="MyPassThroughPoset"/>
```

**Example 9: How a PassThroughPoset can be instantiated in the Hierarchy.xml file**

*MultiPoset and MultiNode and MultiLUBSet*

Tabular data is often represented in a table where the rows represent patient records and the columns represent attributes. A MultiNode is used to represent a data row and an array of MultiNodes is used to represent a table. A MultiPoset is used to find the LUB of one or more rows, as follows:

$$LUB(A,B) = \langle LUB(A_1,B_1), LUB(A_2,B_2), \dots, LUB(A_N,B_N) \rangle$$

Where A and B are two MultiNodes representing two different patient records. i.e. the LUB of a MultiPoset is simply the pair-wise LUB of the constituent elements of its inputs.

To define a MultiPoset, one must first construct Posets of other types such as RangePosets, PowerPosets. These individual Posets are then aggregated into a MultiPoset. A MultiNode in turn is constructed by aggregating Node classes corresponding to the

individual columns. Since the LUB of two MultiNodes may not have a unique value, we introduce the MultiLUBSet class, whose sole data member is an array of LUBSets. Each LUBSet in the array represents the possible generalizations of an attribute.

```

// Assume the code for examples 1,2 and 3 appears above

// create a 3 column MultiPoset
MultiPoset MP = new MultiPoset(3);

// add the nodes
MP.setCol("Age Range",RL,1)
MP.setCol("Has HIV",PL,1)
MP.setCol("Zip",ML,1)
);
// Assume table columns are "age range", "zip", and
// three Boolean fields representing renal failure,
// HIV and colon cancer

// create two MultiNodes
// the first with values <10-30 years, 0213*, (F,T,T)>
// the second with values <20-35 years, 02134*, (F,F,T)>
MultiNode1 MN1 = new MultiNode(3);
MultiNode2 MN2 = new MultiNode(3);

MN1.set(1,(PosetNode) RL1); // Age range: 10-30 years
MN1.set(2,(PosetNode) ML1); // Zip Code: 0213*
MN1.set(3,(PosetNode) PL1); // (False, True, True)
MN2.set(1,(PosetNode) RL2); // Age range 20-35 years
MN2.set(2,(PosetNode) ML2); // Zip Code: 0214*
MN2.set(3,(PosetNode) PL2); // (False, False, True)

LUBSet result = MP.getLubset (MN1,MN2);

// result now holds a LubSet containing a MultiNode
// < [10,35], "021**", (F,T,T)>

```

**Example 10: How a MultiPoset and MultiNode can represent a table**

Unlike other Posets, MultiPosets are never declared in the Hierarchy.xml file. Rather, they are used internally by the toolkit to store rows.

### Columns.XML File

Heretofore we've explained how to create various Posets in the Hierarchy.xml file, but we have not explained how to load data associated with these Posets from a file. Here we show how one can load data from a Comma Separated Values file (CSV) into an array of MultiNodes. But before one can load a CSV file one must first bind one or more Posets to specific columns in the file. This is done via the Columns.xml file.

```
<TableDescription
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance >

  <Column Name="Age_Year" Hierarchy="Default" Poset="Age_range_poset" Weight="3.023" />
  <Column Name="Gender" Hierarchy="Default" Poset="MySuppressionPoset" Weight="988.823" />
  <Column Name="Ethnic_Origin" Hierarchy="Default" Poset="MySuppressionPoset" Weight="18.176"/>
  <Column Name="mrn" Hierarchy="Default" Poset="MyPassThroughPoset" Weight="0"/>
  <Column Name="total_chrg" Hierarchy="Default" Poset="MyPassThroughPoset" Weight="0"/>
  <Column Name="high_exp" Hierarchy="Default" Poset="MyPassThroughPoset" Weight="0"/>
  <Column Name="total_chrg_04" Hierarchy="Default" Poset="MyPassThroughPoset" Weight="0"/>
  <Column Name="other_dx_code" Hierarchy="Default" Poset="SparseMatrixPoset_ICD9"
    Weight="2.566" />
  <Column Name="principal_px_code" Hierarchy="Default" Poset="SparseMatrixPoset_DRG"
    Weight="7.2263" />
</TableDescription>
```

### Example 11: How to use the Columns.xml file to bind Posets to specific columns

As shown in the example above, the columns are declared in same order as they appear in the CSV file. The relative weights for each row can also be specified here. These weights are multiplied by the measure to give the value of any particular data item. At first glance it may seem that the toolkit is founded on weighted indexing but this is not so. While we do provide the facility to implement weighted indexed measures, this facility is optional and left to the discretion of the user. Implementing such a facility also allows us to empirical compare weighted indexes against other measures.

The Hierarchy.xml file allows one to define namespaces, so that multiple Posets could be, for example, defined with the same name. By specifying Hierarchy= "Default" in each column we are declaring that the Poset specified can be found within the default namespace.

## Ambiguators

An ambigator is a class that implements an algorithm for transforming a table stored in a DataSet class into a table that has k-anonymity. We use the terms like *ambiguate* and *ambigator* rather than *anonymize* and *anonymizer* because *k*-anonymity does not always lead to anonymity (as shown on page 24).

The DSG toolkit is based on the Vinterbo optimization framework for anonymization. All k-anonymization algorithms must implement the Ambigator class and receive their data in the form of a DataSet class – a class that simply contains an array of MultiNodes (which represents an array of rows) and a generalization hierarchy of MultiNodes.

```
public class DataSet
{
    public MultiPoset MP;
    public MultiNode[] data;
    :
}
```

Figure 9: Data members of the DataSet Class

Each Ambigator returns its results as an AmbigRun object whose basic data members are listed in Figure 10.

```
public class AmbigRun
{
    // The original data set that was ambiguated
    public DataSet origData;

    // Ambigator used to ambiguate this data
    public Ambigator ambig;

    // Final data --- filled in by Ambigator
    public MultiLUBSet[] ambigData;
}
```

Figure 10: AmbigRun Member Functions

The DSG Toolkit contains three ambiguitors: *Greedy*, *Greedy-DXCG* and *Partition* and the toolkit is extensible to allow for the addition of other algorithms. Each Ambiguator must implement the Ambiguator class. The basic data members of this class are listed in Figure 12. We have abstracted the arguments to each ambiguitor as an AmbigArgs class. This abstraction enables a series of ambiguitors to be called using the same data in a loop (enabling side-by-side comparisons of performance). We now explain in greater detail the built in algorithms.

### *The Greedy Algorithm*

The Greedy ambiguitation algorithm was written by the author. It is essentially a hill climbing algorithm. Given a row, it creates a LUBSet containing the row, and successively adds rows that least increase the measure of the LUBSet until the LUBSet contains  $k$  rows. To ambiguitate an entire table, it simply repeats the steps above for each row. The pseudo-code for the Greedy algorithm is shown in Figure 11.



```

ALGORITHM NAME: Greedy
INPUTS: k Value representing desired level of k-
            anonymity
            MP A MultiPoset.

            LUB(X,y,MP) A function, that returns the least upper
            bound of the set of rows X and the row y
            according to the Poset MP.

            Measure(X) A function which returns the information
            value of the set of rows X.

RETURNS: items[] an array of sets where items[i] contains
            the k integers representing the row
            number to be lubbed with row i.

BEGIN
    rows  $\leftarrow$  [1,N]
    cols  $\leftarrow$  [1,k-1]
    for  $\forall i \in$  rows {
        items[i]  $\leftarrow$  {i}
        for  $\forall j \in$  columns {
            row_to_add  $\leftarrow$  argmin  $t \in$  rows $^{t \neq i}$  measure(LUB(items[i],t)
            items[i] =  $\leftarrow$  items[i]  $\cup$  {row_to_add}
        }
    }
    return items
END

```

**Figure 11: Greedy Algorithm Pseudo-code**

### *The Partition Algorithm*

The partition algorithm was written by Robert Fischer. It finds clusters of size at least  $k$ . It functions by recursively splitting rows of a table into two partitions. Initially the whole data set is regarded as one partition. The algorithm then repeatedly replaces each

partition with two new partitions that were created by splitting the original partition. The process by which a partition is split is outlined as follows. First, the two elements that are “farthest apart” from one another are identified. The distance metric used for determining how far apart two nodes, A and B is as follows:

$$\text{distance} = 2 * [\text{Measure} (\text{LUB}(\text{A},\text{B}))] - [\text{Measure}(\text{A}) + \text{Measure}(\text{B}) ]$$

Where Measure(·) is some function that satisfies the properties defined on page 26. The two nodes found to be farthest apart respectively represent the first elements of the two initial partitions. The remaining elements are then assigned to the partition closest to them. This process is recursive. That is, the partitions created by this process are also split into still smaller partitions, and the process repeats until partitions of size less than  $k$  emerge, in which case the last split is reversed.

#### *The Greedy-DXCG Algorithm*

The Greedy-DXCG algorithm is similar to the greedy algorithm but it uses a different measure and has some specific optimizations. Like the greedy algorithm, it anonymizes a row by turning it into a LUBSet and by successively adding rows that least increase the measure of the LUBSet until the LUBSet contains  $k$  rows.

But unlike the greedy algorithm it contains some special optimizations. These optimizations were necessary because an API version of the DxCG software was not available at the time of the experiment<sup>2</sup>. Whereas the greedy algorithm would call the measure function  $n!/(n-k)!$  times, the Greedy-DxCG algorithm is optimized to only spawn the DxCG process  $k$  times –significantly reducing the overhead of creating and destroying the DxCG process. This reduction in calls to the DxCG software was achieved by inputting data to the DxCG function in batches – not by reducing the number of rows DxCG had to process. Nonetheless, it resulted in significant speed gains.

Another difference between the Greedy algorithm and the Greedy-DxCG algorithm is their flexibility in using different measures. Whereas the Greedy algorithm can use a variety of measures, the Greedy-DxCG is tied to a single measure; namely, the error

---

<sup>2</sup> It's the author's understanding that an API version is planned but not yet released.

between DxCG's predicted costs and the patient's actual costs in year 2. As a result, this algorithm cannot be used on tables except those that have year 1 patient data, and a column containing actual year 2 patient costs. Thus, the Greedy-DxCG algorithm can only be considered to be a specialized algorithm. Nonetheless, we created it to serve as a "gold standard" against we can measure other algorithms, and measures.

### **Synthetic Data Generation**

Included in the DSG Toolkit are tools to generate synthetic data from distributions. Our method of generation is fairly primitive: we randomly generate the data independently of each other based on distributions found in the US census; however, more advanced techniques for synthetic data generation (such as the technique known as "multiple imputation") could possibly be implemented in the toolkit by future researchers.

### **Command Line Interface**

The DSG toolkit has a command line interface that makes it easy to create, generate, evaluate, or ambiguate data. The command line tool is termed "Lubber" – a reference to the LUB operation. The command line options are long and extensive. We do not list them here, but the interested reader can find them by typing "java -jar privacyToolkit.jar -help".

```

public class AmbigRun
{
    // The original data set before ambiguation
    public DataSet origData;

    // Ambiguator used to ambiguate this data
    public Ambiguator ambig;

    // Ambiguated Data
    public MultiLUBSet[] ambigData;

    // An array of k integers for each row that explains
    // which rows were combined to create each row.
    // i.e. ambigData[i] = LUB(items[i][0], ... ,items[i][k])

    public int[][] items;
    :
}

```

**Figure 12: Data Members of the AmbigRun class**

## Experiments

### Context

We wish to simulate a situation where a variety of hospitals will contribute patient data to a hypothetical researcher who is seeking to construct a model that can be applied in a wide variety of geographies and hospitals.

The latter definition has a number of implications. A model that is applicable across a wide variety of geographies and hospitals will typically only use inputs (such as age, gender, and diagnoses) that are hospital or geography independent.

As a result, we believe our typical researcher would not be interested in fields that are geography-specific (such as a zip code) or hospital-specific (such as “name of admitting physician”) unless such fields are first converted into a field that is neither hospital nor

geography specific. An example of such a conversion might be converting the geography-specific field, “zip code” to the more general field “cost-of-living adjusted income.”

Following the above logic, we have chosen to exclude hospital-specific or geography-specific fields from our analysis. This is not to say that such fields are not useful to some researchers. But rather a reflection that such fields may not be of use to a researcher trying to create a risk model of risk for a disease. Moreover there is utility in our choice – by excluding such fields we are likely to increase the information content of geography and hospital independent fields like gender, age and diagnosis.

To simulate a real application for the data, we have outputted our anonymized data into RiskSmart 2.0 – a predictive model produced by DxCG Inc., used by health insurers to predict their expenditures in future years. In particular, the RiskSmart model allows one to predict the cost of each patient in future years, based on a patient’s demographic profile and diagnoses in the current year. Health insurers typically use RiskSmart to identify patients whom they could select for preventive treatment – a procedure often termed “disease management.” The idea behind disease management is simple: “an ounce of prevention can save a pound of cure.”

The RiskSmart model has a number of outputs. For the purposes of this dissertation we have elected to consider only its prediction of 2004 costs based on our 2003 data. For all of our experiments, we measured the performance of the final output on DxGG – by summing the absolute errors between DxCG’s predictions of expenses and the true expenses for each individual.

## The Data Set

We began with real data set consisting of the demographic data, procedures and diagnosis of inpatients from hospitals in the Boston area. Institutional Review Board (IRB) approval was obtained for this study.

The data consisted of the following tables:

- *demo\_03 and demo\_04*: which contain the demographic information of 28,795 and 32,307 inpatients respectively for individuals registered at the health network in the years 2003 and 2004. The fields in these tables are show in Figure 13.

Name	Description
adm_dt1	Admission date
last_name	Last name
first_name	First name
ssn	Social security number
dob	Date of birth
age_day	Days of age over and above that specified in month_age
age_month	Months of age over and above that specified in age_year
age_year	Years of age
Gender	Gender
ethnic_origin	Ethnic Origin expressed as an Integer
race_full	A Full-text representation of ethnic origin
religion	Religion
language	Preferred language of correspondence
marital_status	Marital status
zipcode	Zip code of residence
mrn	Medical records number – a unique identifier for each patient
tot_chrg	Total cost of treating the patient for the year

**Figure 13: Fields in Tables Demo\_03 and Demo\_04**

- *dx\_03 and dx\_04*: these tables contain 932,657 and 1,121,264 diagnoses of inpatients for the years 2003 and 2004 respectively. Entries in this table are linked to those in the demographics table through the mrn field – a unique identifier for each patient. Figure 14 shows the fields for this table.

Name	Description
adm_dt	Admission Date
disch_dt	Discharge Date
enc_num	Encounter Number – a unique ID for the entire hospital stay (i.e. from admission date until discharge)
Mrn	Medical Record Number – a unique ID for each patient
Los	Length of stay in days
Gender	Gender
Zipcode	Zip code of mailing address
Uid	User identifier – another unique identifier for each patient
principal_px_cd	Principal procedure code (a DRG code)
principal_px	The above principal procedure code represented in human readable text.
adm_md	Admitting Doctor
other_dx_cd	Other Diagnosis Code (an ICD-9-CM code)
other_dx	The above diagnosis code represented in human readable text.

**Figure 14: Fields in tables dx\_03 and dx\_04**

As shown in Figure 15, there were only a handful of unique diagnoses.

Year	Rows in Diagnosis Table	Unique Diagnoses	Unique Diagnosis Expressed as a % of total
2003	932,657	382	0.041%
2004	1,121,264	1600	0.143%

**Figure 15: There are only a handful of unique diagnoses**

## Approach

To measure information loss in our experiments we propose a variety of measure functions. These measure functions could be regarded as constituting a spectrum of defensibility ranging from measures that value all fields equally to those that value each data item based on its predicted information content.

### **Method 1: Equal weights** (all columns have the same value of 1.0)

This measure function has two parts. The first computes a score for each field, the second sums the first measure into a score for each row.

This measure function assigns a score of 0 to all leaf nodes, 1 to each leaf's parent, 2 to the leaf parent's parents, and so on and so forth. Each of these values, however are divided by  $M$  – the height of the highest height in the hierarchy. This ensures that the scores range from 0 to 1.0 – 0 being assigned to the leaves and 1.0 being assigned to the topmost node in Poset or DGH.

The measure for each row is simply the sum of the scores for each field in the row. This can be interpreted as assigning a weight of 1.0 for each column. In sum this measure function can be written as

$$Measure(R) = \sum_i \frac{h_i}{M_i}$$

Where:

- $R$  is a row consisting of  $i$  columns
- $h$  is the height of the element in column  $i$  within the Poset or DGH corresponding to column  $i$ .
- $M$  is the highest height in the DGH or Poset corresponding to column  $i$  or the highest possible range in a RangePoset.

This measure function has properties highly similar to the PREC metric discussed in Figure 4 insofar as the position of a field in a DGH or Poset completely determines its score. Such measures would assign the same



score, for example, to knowing if someone had HIV to knowing if they were male or female.

**Method 2: Using empirically estimated column weights**

Whereas the last measure weighted all columns equally, this measure function assigns a unique weight to each column according to an empirically derived measure (the mutual information of that column to the variable we are trying to predict). This empirical measure seeks to estimate the “true” information the column has with the predicted variable. The greater the amount of reliable data one has to estimate the mutual information, the greater the confidence one can have that the estimate of mutual information converges with its true score.

$$Measure(R) = \sum_i \frac{h_i}{M_i} w_i$$

Where  $w_i$  is defined as the mutual information between column  $i$  and the column we are trying to predict. The mutual information between a column  $X$ , and a column we are trying to predict  $C$ , is defined as:

$$MI(X;C) = \sum_i \sum_j p(X_j, C_i) \log_2 \left( \frac{p(X_j, C_i)}{p(X_j)p(C_i)} \right)$$

The mutual information is an information theoretic measure. It represents the decrease in entropy of the random variable  $C$ , given the random variable  $X$  i.e.:

$$MI(X;C) = H(C) - H(C|X)$$

### **Method 3: Using an empirically estimated information measure of each value**

Unlike the latter measure which assigns a single number to an entire column, this measure assigns a value to each cell within a table. This assigned value is the point-wise mutual information with the target variable.

The point-wise Mutual Information between a vector  $x$ , and a target variable  $C$  (the column we are trying to predict) is defined as:

$$PMI(X = x, C = c) = \log_2 \left( \frac{p(x, c)}{p(x)p(c)} \right)$$

If the values  $x$  and  $c$  are statistically dependent, the mutual information will be positive. Conversely, if the two values are disassociated greater than chance, the Mutual Information will be negative.

We have chosen to value both evidence of association and disassociation equally by using the absolute value of the point-wise mutual information as our measure. Further, because our system seeks to minimize scores we multiplied the absolute value of the score by -1 – so data values with more information will be treated as more valuable.

There are some issues with this measure. First, by summing the pointwise mutual information scores for each column of a row into a single measure, it assumes that the columns are all independent – i.e. that there is no redundancy in the information between the columns – an assumption almost certainly violated in practice. Second, it violates the monotonicity property we required in a measure function earlier because a generalized value may actually have a higher score than its ancestors. This discrepancy reflects the fact that our mutual information estimate is exactly that: an estimate. Had our tables had enough rows, our values would converge to the true values and monotonicity would be preserved. We therefore relax our requirement for monotonicity when using this measure.

**Method 4: Using the degradation of the predictive model we intend to use as our measure**

This measure is perhaps the most rationally defensible. It uses the degradation in predictive performance in our intended application as our measure of information loss. Unlike the other methods, it escapes the weighted index problem by considering a row in totality rather than by summing individual measures for each column.

For the purposes of our experiments our final intended predictive model is DXCG Inc.'s RiskSmart 2.0.

**How we calculate mutual information scores:**

To the tables dx\_03 and demo\_03 we added a column called "high\_expense." This column holds a Boolean value that was set to 1 if a patient had expenditures of more than \$25,000 in year 2 (i.e. 2004), and 0 otherwise. This is the target variable for all mutual information calculations. We also added implied rows to form an augmented table as describe in Figure 7. This allowed us to calculate a mutual information score for each level of generalization in all Posets.

Mutual information scores are known to be biased towards rare values. For the sole purpose of calculating mutual information scores we temporarily removed all values occurring less than 6 times.

This resulted in the following:

- a. 405 zip codes remained out of 1537
- b. One age\_year (14) was removed (it occurred only once)
- c. One ethnic group race was eliminated (NATIVE HAWAIIAN which occurred only once)

A Perl script was used to calculate these mutual information scores. Its output was portions of the Hierarchy.XML configuration file.

## **Data Cleansing and Data Preparation Procedures**

As is typical of most data sets, we discovered a number of inconsistencies in our data. For example, we noticed that a number of individuals had their age\_year field set to zero – even though their dates of birth were in the distant past.

In order to prepare our data for anonymization we performed a number of procedures:

- 1. Removed obvious identifiers** – including first name, last name, uid, social security number, and address.
- 2. Removed redundant fields** – a field is redundant relative to another if knowing the second field adds no information not already known from the first. For example, the field “race\_full” adds no information if one already knows the value of the field “ethnic group.” The former is a number representing an ethnic group, the latter, merely a human readable full text version of the first. This resulted in the elimination of the field race\_full. We also deleted the field principal\_px which was simply a human readable version of the ICD-9-CM code stored in principal\_px\_cd and other\_dx\_cd which was a human readable version of the DRG code stored in other\_dx. We also elected to delete the fields admission date and discharge date since they had a very low mutual information score and much of their predictive information seemed to be captured in the field los (which stores the length of stay in the hospital).
- 3. Set all ages  $\geq 90$  to 90+** – this is a common practice and is consonant with the procedures used to anonymized records in HIPAA.
- 4. Deleted patients in demo\_03 and dx\_03 2003 that did not exist in demo\_04** – To test our anonymized data, we programmed DxCG’s RiskSmart software to predict the cost of a patient in 2004 based on their profile in 2003. But unless we have a patient’s 2004 data (including their actual 2004 expenses), we have no way of calculating the error between the predicted and actual costs. Thus for the purposes of our experiment, we deleted all patients who didn’t have both 2003 and 2004 data.

5. **Deleted hospital-specific and geographic specific information** – This is in line with our assumption outlined above. Fields purged include zipcode and admitting physician.
6. **Deleted data items with low mutual information ratios** – Mutual information scores are known to be biased towards elements with a large number of possible values. To counteract this bias, we divided each mutual information score by the mutual information score of a column with random values chosen from a set with the same number of possible values. This ratio may be termed the *mutual information gain* or *MI Ratio* for short. We used the MI ratio as a comparison of information content across fields (see Appendix 1 for a proof that Mutual Information scores can be compared across tables). To estimate the MI ratio we computed the value 10 times, and took the average of the 10 runs.

We chose to delete all entries with a mutual information gain of less than 2.0. This threshold (two times random chance) was arbitrarily chosen. Our choice was somewhat supported by the fact that age\_year had a ratio just above three while age\_month and age\_day – fields that one would expect would essentially be random and have little medical causal effect – had MI ratios of 1.73 and 0.83 respectively; thus placing our threshold above random chance, but below a value known to have medical causality. We also eliminated enc\_num – a unique ID assigned to each patient visit. Figure 16 shows the fields that were kept and discarded following this procedure.

Field Name	Unique Values	MI	MI-random	MI Ratio	Used by DxCG?	Kept
enc_num	4334	0.635	0.423	1.5	No	No
adm_dt1	365	0.077	0.070	1.1	No	No
disch_dt1	393	0.078	0.071	1.1	No	No
Ssn	4296	0.630	0.418	1.5	No	No
Dob	3961	0.576	0.421	1.4	Yes	No
age_day	31	0.006	0.007	0.8	No	No
age_month	12	0.003	0.001	1.7	No	No
age_year	71	0.038	0.012	3.1	Yes	Yes
Gender	2	0.010	0.000	212.7	Yes	Yes
ethnic_origin	9	0.018	0.001	21.0	No	Yes
Religion	16	0.004	0.004	1.1	No	No
Language	22	0.004	0.004	1.2	No	No
marital_status	7	0.005	0.003	1.9	No	No
Zipcode	519	0.107	0.105	1.0	No	No
Mrn	4334	0.635	0.446	1.4	No	Yes
tot_chrg	4207	0.623	0.433	1.4	Yes	No
Los	109	0.0237	0.022	1.1	No	No
Tot_chrg_04	4216	0.635	0.426	1.5	No	Yes

**Figure 16: Fields that were kept and discarded in demo\_03**

In some cases, the same field would appear in both the demographic and diagnoses tables. In each case, we had to determine which table was best to estimate the MI\_ratio for that field. In the case of demographic fields such as gender, it would be incorrect to estimate their MI\_ratio from the dx\_03 table because the same person will have the gender repeated many times – in effect giving greater weight to people with more diagnoses.

Field Name	Unique Values	MI	MI-random	MI Ratio	Used by DxCG?	Kept
adm_dt	365	0.01511888	0.00256879	5.8856	No	N/A
Disch_dt	393	0.0167427	0.00278111	6.02014	No	N/A
Enc_num	32470	0.63772793	0.23930418	2.66493	No	No
Mrn	4332	0.63772793	0.03251145	19.6155	Yes	Yes
Gender	2	0.00472171	5.06E-06	933.58	Yes	N/A
Principal_px_cd	817	0.05547713	0.00614004	9.0353	No	Yes
Other_dx_cd	3787	0.06422494	0.02948036	2.17857	Yes	Yes

**Figure 17: MI Ratios and fields kept in Dx\_03. An “N/A” in the “kept” column acknowledges the fact that the MI could not be appropriately estimated from this table.**

The final result of our data preparation and data cleansing procedures was a table with two extraneous fields – extraneous to the extent that these fields were not used by the DxCG predictive model. It should be noted, however, that the single extraneous field, principal\_px\_cd (principal procedure code) will later be shown to translate into 15 separate fields in the final table used for the anonymization process. This could potentially amplifying the extent of the “damage” this single field could bring to the anonymization process insofar as the presence of extraneous fields in the anonymization process are likely to divert the ambiguation algorithm from preserving data from the fields that will actually be used (in the final predictive model).

7. **Deleted data items occurring fewer than k times** – On page 25, we outline an attack that can be applied to data items occurring fewer than  $k$  times. To thwart this attack all data items occurring fewer than  $k$  times in the data set were deleted. For the purpose of this experiment we have set  $k$  to 3.
8. **Preserved fields needed to calculate error** – we configured our anonymization software to “pass-through” certain fields whose sole purpose was to facilitate calculate error. To ensure such fields had no effect on the anonymization process,

we set the weight of such fields to zero, and used a “PassThroughPoset.” Fields in this category include `mrn` (a unique medical records identifier), `tot_chrg_04` (the patient’s total expenses in 2004) and `high_expense` (a Boolean field set to 1 only if a patient’s expense in 2004 > \$25,000 and 0 otherwise).

Finally, following these data cleansing and preparation procedures, we randomly selected 1000 patients and joined their information in the `demo_03` and `dx_03` tables as follows. First we noted that a patient had at most 62 distinct diagnoses. However only 17 people had more than 39 diagnoses – allowing us to completely capture the diagnoses of  $983/1000 = 98.3\%$  of the patients in the data set using two groups of 39 columns for each person. These two groups represented “name-value” pairs. The first 39 columns (Called `dxFieldName1`, `dxFieldName2`, ... , `dxFieldName39`) stored the diagnosis codes (i.e the names). While the remaining 39 columns (named `dxFieldVal1`, `dxFieldVal2`, ... , `dxFieldVal39`) stored the corresponding frequencies for each diagnoses (i.e. the values). The diagnoses were stored from left to right. If a patient did not have as many as 39 diagnoses the rightmost fields would hold one or more blank values. If a patient had more than 39 diagnoses, the latter diagnoses were purged; however, this only happened in 1.7% of the cases. These two groups of 39 columns were then loaded into a `SparseMatrixNode`, to enable the LUB operation to be performed on them across rows. Figure 18 shows the final table used for our experiments.



Field Name	Unique Values
age_year	71
gender	2
ethnic_origin	8
mrn	1000
tot_chrg	977
high_exp	2
tot_chrg_04	982
dxFieldName1	263
dxFieldVal1	21
⋮	⋮
dxFieldName39	13
dxFieldVal39	4
pxFieldName1	222
pxFieldVal1	31
⋮	⋮
pxFieldName15	5
pxFieldVal15	4

Figure 18: The final table used for experiments

## Experimental Setup

1. We seek to answer three questions:

a) **Which algorithm is best? Greedy or Partition?**

b) **Which measure function is best? Vinterbo-1.0, Column-MI or Value-MI?**

We have defined three measure functions to guide our anonymization process (which essentially constitute an optimization problem). Under what circumstances are any of the measures better than the others? Does the choice of anonymization algorithm or size of the data set affect the outcome?

c) **Can we predict what fields will ultimately be used by the end user to predict the target variable in their models? To what extent do mistakes in**

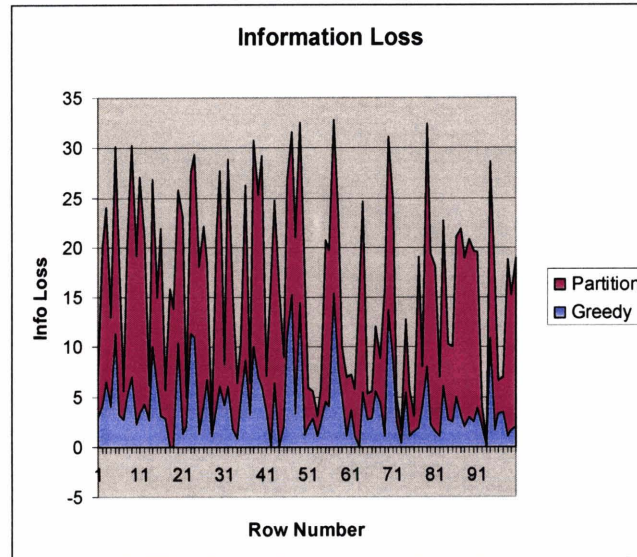
**predicting the fields used by the end-user affect the predictive performance of the ultimate model?**

Most predictive models take as input a certain set of pre-specified fields. The same is true of our anonymization process. While we may not be able to anticipate the variables used by researchers to build their predictive models, we do have control over what fields we wish to keep or purge before we even begin the anonymization process. Our choices of which fields to keep and which to discard can have an affect on the accuracy of the end user's predictive model. That is, if we input extraneous fields (extraneous in the sense that they are not ultimately used by the end-user), one would expect the anonymization algorithm to sacrifice some precision in these fields to preserve information in the extraneous fields. Can we successfully predict the fields that will be needed by our model? And if not, will mistakes in field selection matter?

We seek to answer these questions by conducting a number of experiments. First, we compare greedy to partition on synthetic data. Second we compare greedy to partition using various measure functions on real patient data. Third we compare the performance of these algorithms on data sets of different sizes.

a) Performance of greedy and partition on synthetic data

A test set of 100 rows was randomly generated based on census data for Hampshire country, Massachusetts. The fields generated included a census block (the most finely granular physical location available in the census data as shown in Figure 3), Age (again based on census data), and 5 randomly generated Boolean fields (the first which could be interpreted as setting the randomly generated person as male or female), the remaining 4 being randomly generated and could be thought to simulate the presence or absence of a disease or condition. A Vinterbo style measure function was used (where the value of cell is *represented* by its hierarchy and all columns are weighted equally).



**Figure 19: Greedy clearly outperforms Partition on synthetic data**

Figure 19, shows that greedy clearly dominates the partition algorithm according to the Vinterbo measure of information loss. This would seem to indicate that Greedy outperforms partition on data sets where the fields are generated independently of one another. However, in the next experiment we'll show that on real data, where there does exist dependence between fields, Partition clearly outperforms Greedy as measured by the predictive performance of the outputted data in our final application (DxCG).

In our next experiment we used the dataset of 1000 patients from the 2003 dataset described earlier. This dataset was anonymized using the greedy and partition algorithms that were run using 3 different measure functions.

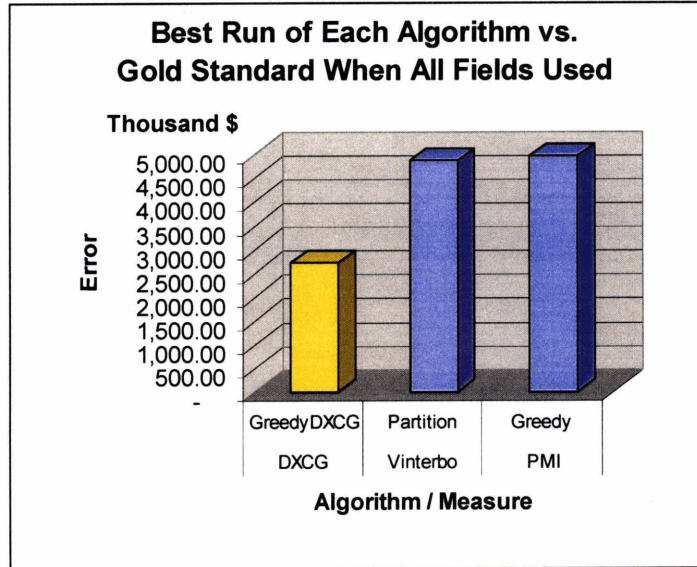
1. Vinterbo-1.0 – where every field is assigned a value according to its position in the hierarchy for its respective column. The score for a row is simply the sum of scores for each column. We add the designation “1.0” to the name of the measure function because it sums scores for each columns with a weight of 1.0; and it is only one example of the allowable functions in the Vinterbo framework; however for brevity, all references hereafter to this measure will simply be referred to as “Vinterbo.”

2. Col-MI – similar to the Vinterbo measure, however each column is assigned a weight corresponding to the mutual information between that column and the target variable – in this case high\_expense (a Boolean variable which is 1 if a patient’s expenses exceeded \$25,000 in 2004, 0 otherwise). To calculate the value of a row, one calculates the values for each column as per Vinterbo, but each column value is multiplied by its corresponding weight before the sum is taken.
3. Value-MI – where every distinct value in a column is assigned a score according to its point-wise mutual information with the high\_expense field.

The output of the anonymization process was then fed through a Perl script, which did a number of operations to prepare the data for input into DxCG’s Risk Smart model:

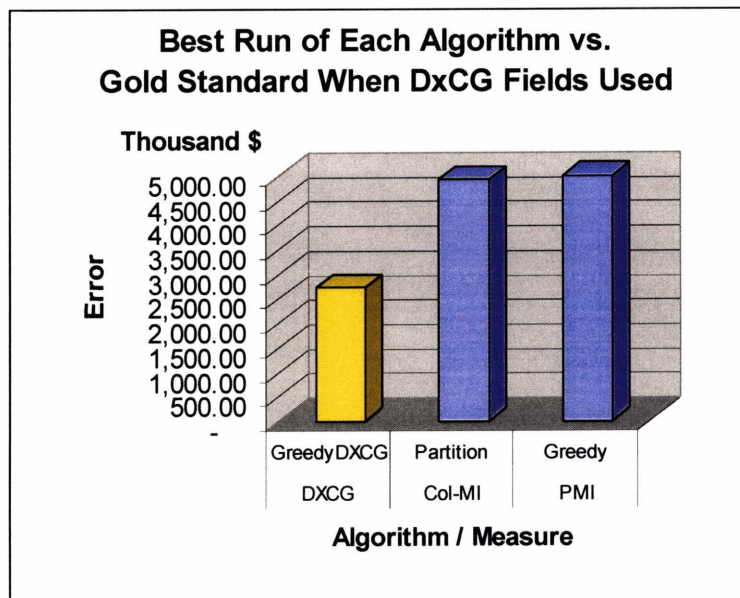
- a) Deleting fields not used by the DxCG model (including ethnic group, and principal\_px\_cd)
- b) Converting all age ranges (such as [10,20]) into the mean of the range.
- c) Duplicating every row with a suppressed gender into two fictitious persons: with all attributes identical to the original person – except that the first person was assigned the gender male, the second assigned female. This step is necessary because DxCG cannot handle individuals with a suppressed gender. Our way of overcoming this was to input both genders into DxCG and to take the average of the predicted expenses of the two fictitious persons.
- d) After the data was prepared it was passed through the DxCG model which was programmed to output a prediction of a patient’s 2004 expenses based on their profile from the previous year (2003). The “error” for each individual was computed as the difference between a person’s actual expenses in 2004 and DxCG’s predicted expense. The sum, mean, and standard deviation of all such errors were taken.

As shown in Figure 20 and Figure 21, the partition algorithm outperformed greedy both in the presence and absence of extraneous fields.



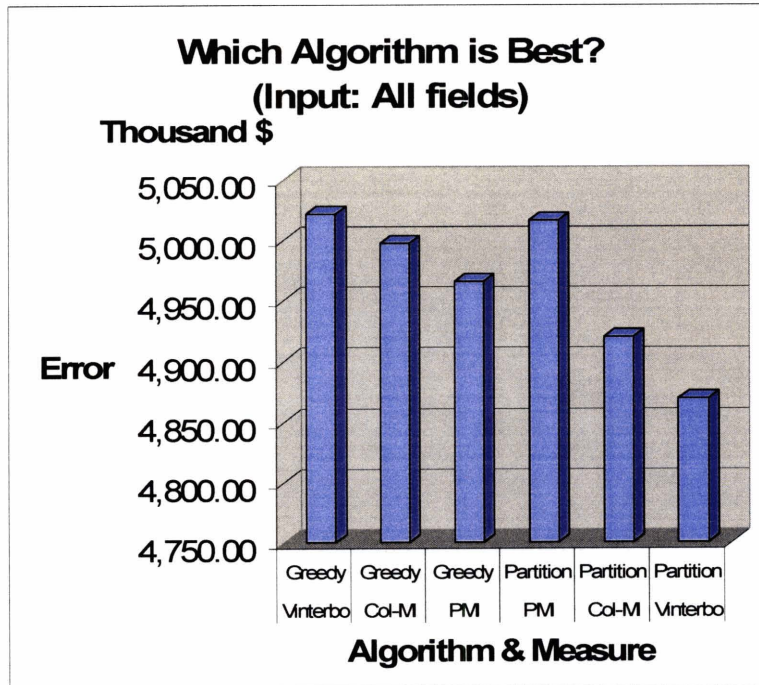
**Figure 20: Best run of each algorithm when all fields used**

As shown in Figure 21, Partition using the Vinterbo measure outperformed greedy when extraneous fields were inputted into the anonymization process.



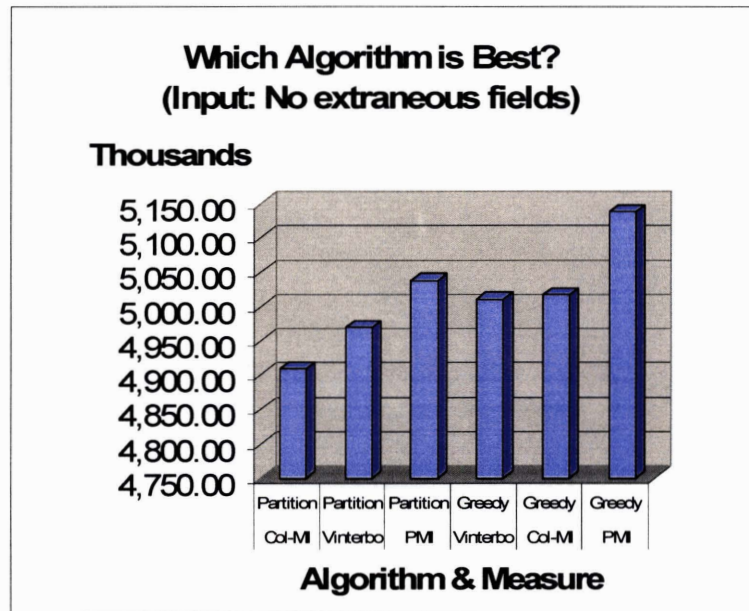
**Figure 21: Best run of each algorithm when no extraneous fields were used**

However we notice an interesting result in Figure 22 – that there is no best measure



**Figure 22: Comparison of Greedy with Partition when all fields including extraneous ones were used.**

per se. Rather, the measure function that is “best” is highly dependent on which algorithm was used. In fact, a complete rank reversal was observed; the best measure function for the greedy algorithm (Vinterbo) was the worst measure function for Partition – and vice versa.



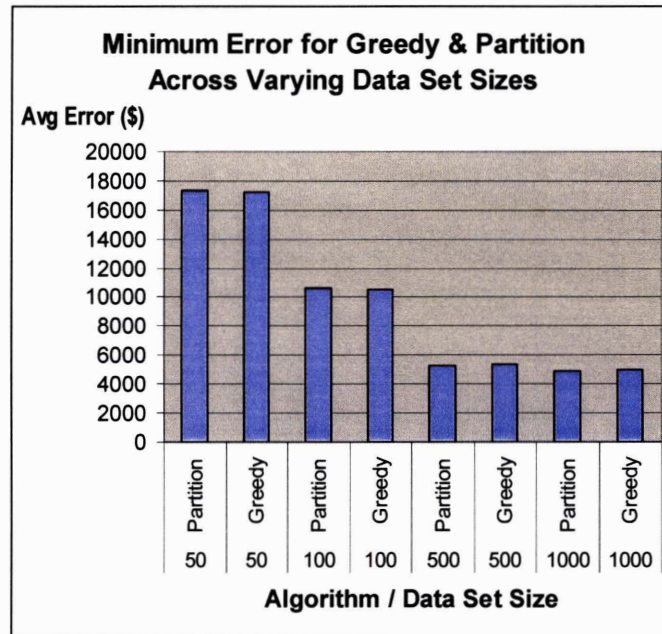
**Figure 23: Comparison of Greedy to Partition when no extraneous fields were used**

The presence of extraneous fields had a marked effect on our results. When extraneous fields had been removed, the best measure function for partition was the Col-MI measure whereas in the presence of extraneous fields, best performance came from the Vinterbo measure.

Upon reflection this result was not surprising because the extraneous fields (i.e. principal\_px\_cd and ethnic\_group) had higher mutual information scores than a critical field used by DxCG called “other\_dx\_cd,” which stores the ICD9-CM codes associated with the patient.

This meant that the Col-MI measure function de-emphasized a critical field for DxCG while simultaneously applying a higher weight to extraneous fields. It is therefore not surprising, that the Vinterbo measure (which gives equal weight to all fields), outperformed COL-MI in the previous experiment because the Col-MI measure served to amplify the negative impact of the extraneous fields whereas the Vinterbo measure did not. Conversely, when all extraneous fields were removed, one would expect Col-MI (which recognizes the relative weights of columns) to outperform the Vinterbo measure. This was also the case.

So far, the partition algorithm has always outperformed the greedy algorithm. Might the size of the data set affect the outcome? To investigate, we repeated our initial experiment (which was conducted on a data set of 1000 rows) on data sets of 500, 100, and 50 rows. These smaller data sets were created by truncating the original at the desired number of rows from the one end of the file.



**Figure 24: Average Error Across Different Data Set Sizes (Input: All fields)**

As shown in Figure 24, the average error is inversely related to the data set size regardless of whether partition or greedy is used. This is not surprising because the smaller the data set, the less likely the algorithm will find  $k$  records that are substantially similar. Thus, in small data sets, the algorithms are forced to combine individuals with little in common – resulting in the wholesale data loss. But as the data set size grows, the likelihood of finding similar individuals increases – enabling such individuals to be combined with less information loss.

Varying the size of our data sets also allows us to analyze the data in another important way: it allows us to see if the best measure for each algorithm was consistent across data sets of varying sizes. This in turn enables us to provide some recommendations for the best measure for each algorithm in a more rigorous fashion than our previous



analysis – which only measured performance on a data set of a fixed size (1000 individuals).

To determine the best measure for each algorithm, we analyzed the performance of the measures by calculating the overall error across all data set sizes and by ranking the performance of the measures in each run, and converting the ranks into “votes” using two voting mechanisms (Johnson 2005):

A) Condorcet Voting (Schulze 2003) – which turns the ranks for each data set size into a vote.

B) Majority Voting (Green-Armytage 2006) – which counts as a vote only the best measure for each data set size.

Before proceeding, we will explain the two voting schemes here.

### **Majority Rule Voting**

This is a very common voting scheme and is the voting scheme used in most political elections. The winner of the vote is the candidate that receives the most votes. In the context of our experiment, each data set size will cast a vote corresponding to the best measure for its data set. Two “elections” were held – one to determine the best measure for the greedy algorithm, the other to determine the best measure for the partition algorithm.

There are, however, some well-known problems with the majority voting scheme. For instance, it is considered the “least-democratic” among commonly used voting schemes – in the sense that it is least likely to reflect the will of voters insofar as it can declare a candidate a winner even though that candidate would have lost against another candidate in a two-way race. This is a reflection of a limitation in this voting system that allows a voter to only specify their “first-preference,” thus disallowing a voter from transferring their vote to a second-preference, should their first preference not win.

### **Condorcet Voting**

This system was invented by the 18th century French mathematician and philosopher Marie Jean Antoine Nicolas Caritat, who held the title of “the Marquis de

Condorcet.” Unlike the prior voting scheme, this voting scheme allows voters to rank candidates. It has been often presented as an alternative voting scheme in political elections – and is considered to have fewer irregularities than majority-voting.

The system works as follows. Voters rank candidates in order of preference. For every possible combination of two candidates, a simulated two-way race is held. In each two-way race, all the votes in each ballot are discarded except the highest-ranked candidate among the two being considered. In this fashion, every ballot can vote for a preferred candidate in every simulated two-way race. The candidate that wins the most two-way races is considered the winner.

For the purposes of our experiment, the Condorcet voting function offers a convenient way to convert the ranking of measures for each data set into a vote that captures such rankings. In particular, every data set size represents a ballot. Each ballot will consist of the ranking of the measures for that data set size. The winner will be the measure that wins the most simulated two-way races.

Best Measure Functions Across Data Sets of Varying Sizes			
When Data Set of Size 50 Was Not Included			
	<b>Condorcet</b>	<b>Majority Win</b>	<b>Total Error</b>
<b>Greedy</b>	Vinterbo	Vinterbo (67%)	Vinterbo
<b>Partition</b>	Col-MI	Col-MI (100%)	Col-MI
When Data Set of Size 50 Was Included			
	<b>Condorcet</b>	<b>Majority Win</b>	<b>Total Error</b>
<b>Greedy</b>	Vinterbo	Vinterbo (50%)	Vinterbo
<b>Partition</b>	Col-MI	Col-MI (100%)	Col-MI
Overall Consensus of The Best Measure Function			
<b>Greedy</b>	Vinterbo	(100% Agreement)	
<b>Partition</b>	Col-MI	(100% Agreement)	

**Figure 25: Performance of Measures Across Data Sets of Varying Sizes When No Extraneous Fields Were Used.** The Vinterbo measure was the best measure for the Greedy algorithm, regardless of what metric was used. Similarly Col-MI was the best measure for the Partition algorithm.

As noted before, for small data sets of randomly selected real patients, k-anonymization is likely to result in wholesale information loss. It is in this vein that we consider the data set size of 50 a special case. For such a small data set, the LUB operator will be likely be highly destructive – even for small k values such as 3. Because we surmise that such small data sets are not representative of the need of real world applications, we have opted to analyze the question of “which measure is best?” both with and without the inclusion of the data set of 50 rows.

As shown in Figure 25, when no extraneous fields were used, this separate analysis made no difference in our conclusions – that is, regardless of which metric was used, the Vinterbo measure was found to be the best measure for the Greedy algorithm, and the Col-MI measure was found to be the best measure for the Partition algorithm – regardless of whether the total error across all runs, the Condorcet voting scheme or the majority win voting scheme was used.

Best Measure Functions Across Data Sets of Varying Sizes			
When Data Set of Size 50 Was Not Included			
	<b>Condorcet</b>	<b>Majority Win</b>	<b>Total Error</b>
<b>Greedy</b>	PMI	PMI	Col-MI
<b>Partition</b>	Col-MI	Tie: Vint, Col-MI, PMI	Vinterbo
When Data Set of Size 50 Was Included			
	<b>Condorcet</b>	<b>Majority Win</b>	<b>Total Error</b>
<b>Greedy</b>	Tie: Col-MI, PMI	PMI (75%)	PMI
<b>Partition</b>	Tie: Col-MI, PMI	PMI (50%)	Vinterbo
Overall Consensus of Best Measure Function (% agreement in parenthesis)			
	<b>Not Incl. 50 Data Set</b>	<b>Incl. 50 Data Set</b>	
<b>Greedy</b>	PMI (66%)	PMI (84%)	
<b>Partition</b>	Tie: Col-MI, Vinterbo (66%)	Tie: Vinterbo, Col-MI, PMI (50%)	

Although the inclusion of the data set with 50 rows made no difference in our results, the same was not true when extraneous fields were added. In particular including the 50-row data set confused our results – creating a 3 way tie for Partition. Interestingly, when no extraneous fields were used the Vinterbo metric was best; however, when extraneous fields were added, the Point-wise Mutual Information (PMI) metric was found to be better. We surmise the reason for this is that the Point-wise Mutual Information metric mitigates to some extent the effect of extraneous columns.

	Algorithm	Measure	Total Error	% Diff from Top	Rank
Including Data Set with 50 Rows	Greedy	Vinterbo	9,642,409	0.301%	3
	Greedy	Col-MI	9,627,222	0.143%	2
	Greedy	PMI	9,613,439	0.000%	1
	Partition	Vinterbo	9,485,802	0.000%	1
	Partition	Col-MI	9,499,280	0.142%	2
	Partition	PMI	9,683,880	2.088%	3
Not Including Data Set with 50 Rows	Greedy	Vinterbo	8,770,461	0.285%	3
	Greedy	Col-MI	8,745,556	0.000%	1
	Greedy	PMI	8,752,661	0.081%	2
	Partition	Vinterbo	8,601,493	0.000%	1
	Partition	Col-MI	8,627,070	0.297%	2
	Partition	PMI	8,816,157	2.496%	3

**Figure 26: Total error for all algorithms across data sets of varying sizes.**

### Experimental Conclusions:

Although one would need to repeat this experiment on many data sets to make strong conclusions, our data suggests the following:

- For data sets where all fields are generated independently of each other (something that rarely occurs in practice):
  - Greedy outperforms Partition
- For real patient data
  - Partition outperforms Greedy
- When the fields that will ultimately be used are known (and are the only fields passed into the anonymization process):
  - The best measure for the *Partition* algorithm is *Col-MI*.
  - The best measure for the *Greedy* algorithm is *Vinterbo*.
- When the fields that will ultimately be used are unknown:
  - The best measure for the Greedy algorithm is *PMI*

- The best measure for the Partition algorithm is Vinterbo. The Col-MI measure came in a close second.

## **Conclusion**

In this dissertation we have presented a new threat model for privacy and a toolkit that allows us to measure the effectiveness of various approaches to achieving anonymity. We have introduced two new general-purpose algorithms for anonymizing data.

From a theoretical perspective, we have also shown that previously published measures of information loss are difficult to defend rationally; while also introducing a variety of measures that in principle are more attractive than previously published measures.

Our new measures represented a spectrum of defensibility – ranging from the least defensible to the most defensible. To see if greater rational defensibility actually made a difference in practice, we empirically tested the performance of our measures on a real application (predicting future healthcare costs) and using real patient data.

We found that the most defensible measure had significantly better performance than less defensible measures; while, our less defensible measures only marginally outperformed their indefensible counterparts.

We have also introduced new theories including the concept of a virtual attack database for precisely modeling privacy threats. And by introducing the concept of augmented tables we were able to show that generalization and suppression; heretofore regarded as distinct operations are in fact the same thing when a table is augmented to include the fields that are implied by the original table.

## **Future Directions / Questions**

There are several promising avenues for our research. The first possible avenue of future research involves harnessing the Bayesian model construction techniques outlined on page 31 to preserve more information in the de-identification process. The theory for this line of research follows.

First we draw attention to the fact that one can infer a probability model from the concept of  $k$ -anonymity. The probability model is as follows. If an anonymized row has  $k$ -anonymity of  $k$ ; then the probability of correctly selecting the person among the list of  $k$ -

possible candidates is  $1/k$ . We call this latter probability the “probability of re-identification.” The reciprocal relationship between the probability of re-identification and k-anonymity is important because one could potential train a Bayesian network to predict the probability of re-identification given any given data row. This is in turn could allow one to train the Bayesian model on a large data set to predict k-anonymity in smaller data sets.

The impact of doing so could be significant because if such a Bayesian network could be constructed and if its error bounds could be understood, one could construct a system that could potentially preserve more information in the anonymization process. In particular, when anonymizing data in a smaller data set, we may no longer need to achieve k-anonymity within the data set if the Bayesian network can predict with strong confidence that the data will have k-anonymity within the larger population. In sum, this line of research may provide the capability to predict k-anonymity of a datum within the larger population.

The second avenue of research would involve testing the data on different kinds of data sets. We have only tested the data on medical data sets. What holds true for medical data, may not hold true for other kinds of data.

A third line of research involves expanding the measure functions included in the toolkit to include other search algorithms such as tabu search, genetic programming, and a greedy search that maintains  $n$  of the top entries (instead of just one).

## Appendix 1: Proof that Mutual Information Scores Can be Compared Across Tables

Are mutual information scores in tables of different lengths comparable?

To answer this question we present the following proof that shows that under certain conditions the mutual information scores between two variables are independent of table length. These conditions are namely that (1) the two variables are generated by the same process in both tables (2) the length of the shorter table is “sufficiently long” to capture the true probability distribution between the two variables and (3) all combinations of the two variables that appear in one table, also appear in the other.

The proof follows.

Let  $T_1$  and  $T_2$  be two arbitrary tables with  $|S_1|$  and  $|S_2|$  rows respectively (we used the variable  $S$  to denote “sample space.”)

Let  $X_1$  and  $X_2$  represent variables in  $T_1$  and  $T_2$  respectively.

Let  $C_1$  and  $C_2$  represent variables in  $T_1$  and  $T_2$  respectively.

Let  $MI(X_1;C_1)$  and  $MI(X_2;C_2)$  respectively represent the mutual information between the random variables  $X_1$  and  $C_1$  in table 1 and  $X_2$  and  $C_2$  in table 2.

Let  $p_1(x)$  ,  $p_2(x)$  respectively represent the probability of the random variables  $X_1$  and  $X_2$  taking on the values  $x_1$  and  $x_2$ .

Let  $p_1(c)$  ,  $p_2(c)$  respectively represent the probability of the random variables  $C_1$  and  $C_2$  taking on the values  $c_1$  and  $c_2$ .

Let  $p_1(x,c)$ , and  $p_2(x,c)$  represent the probability of finding a row with the X-column holding value  $x$  and C-column holding variable  $c$  in tables 1 and 2 respectively.



Let  $N_{x_1, c_1}$  and  $N_{x_2, c_2}$  represent the frequencies of co-occurrence of the column X holding value  $x$  and the Column C holding value  $c$  in tables 1 and 2 respectively.

Let  $N_{x_1}$  and  $N_{x_2}$  represent the frequencies of occurrence of the random variable  $X_1$  and  $X_2$  holding values  $x_1$  and  $x_2$ .

Let  $|S_1|$ , and  $|S_2|$  respectively represent the number of rows in  $T_1$  and  $T_2$

Let  $N_{c_1}$  and  $N_{c_2}$  represent the frequencies of occurrence of the random variable  $C_1$  and  $C_2$  holding values  $c_1$  and  $c_2$ .

Under the following assumptions:

- A1. The pair  $(x_i, c_i)$  is generated by the same random vector  $(X, C)$  in both  $T_1$  and  $T_2$
- A2. Every pairs  $(x_i, c_i)$  is independently generated from the other pairs.
- A3. All combinations of  $(x, c)$  occurring in  $T_1$  also occur in  $T_2$  and vice-versa (note: this will probabilistically hold based on A1 and A2 if  $T_1$  and  $T_2$  contain a “large” amount of rows).

We wish to show that:

$$MI_1(X_1; C_1) / MI_2(X_2; C_2) = 1.0$$

I.e. that the same mutual information score will be arrived at in both tables regardless of the length of each table.

Proof:

$$\frac{MI_1(X_1;C_1)}{MI_2(X_2;C_2)} = \frac{\sum_{x_1,c_1} p(x_1,c_1) \log_2 \frac{p(x_1,c_1)}{p(x_1)p(c_1)}}{\sum_{x_2,c_2} p(x_2,c_2) \log_2 \frac{p(x_2,c_2)}{p(x_2)p(c_2)}}$$

$$\frac{MI_1(X_1;C_1)}{MI_2(X_2;C_2)} = \frac{\sum_{x_1,c_1} \frac{N_{x_1,c_1}}{|S_1|} \log_2 \frac{\frac{N_{x_1,c_1}}{|S_1|}}{\frac{N_{x_1}}{|S_1|} \frac{N_{c_1}}{|S_1|}}}{\sum_{x_2,c_2} \frac{N_{x_2,c_2}}{|S_2|} \log_2 \frac{\frac{N_{x_2,c_2}}{|S_2|}}{\frac{N_{x_2}}{|S_2|} \frac{N_{c_2}}{|S_2|}}}}$$

$$\frac{MI_1(X_1;C_1)}{MI_2(X_2;C_2)} = \frac{\frac{1}{|S_1|} \sum_{x_1,c_1} N_{x_1,c_1} \log_2 \frac{|S_1| N_{x_1,c_1}}{N_{x_1} N_{c_1}}}{\frac{1}{|S_2|} \sum_{x_2,c_2} N_{x_2,c_2} \log_2 \frac{|S_2| N_{x_2,c_2}}{N_{x_2} N_{c_2}}}}$$

Let  $t = |S_2|/|S_1|$  represent the ratio of lines in  $T_1$  and  $T_2$ . Thus we have:

$$\frac{MI_1(X_1;C_1)}{MI_2(X_2;C_2)} = t \cdot \frac{\sum_{x_1,c_1} N_{x_1,c_1} \log_2 \frac{|S_1| N_{x_1,c_1}}{N_{x_1} N_{c_1}}}{\sum_{x_2,c_2} N_{x_2,c_2} \log_2 \frac{|S_2| N_{x_2,c_2}}{N_{x_2} N_{c_2}}} \quad (1)$$

By assumption A1, we know that the values for columns X and C were generated by the same process and by A2, we know the pairs are independently generated. Based on the latter two assumptions, the occurrence of the pairs is a function of

length. Thus for a sufficiently long<sup>3</sup>  $T_1$  and  $T_2$  we can expect that any frequency of occurrence in  $T_1$ , can be converted to its counterpart in  $T_2$  by multiplying by a constant; i.e. that:

$$N_{x_2, c_2} = t \cdot N_{x_1, c_1} \quad (2)$$

$$N_{x_2} = t \cdot N_{x_1} \quad (3)$$

$$N_{c_2} = t \cdot N_{c_1} \quad (4)$$

Substituting (2),(3) and (4) into (1) and noting that by A3, the summation in the numerator covers the same pairs of values as that in the denominator we have:

$$\frac{MI_1(X_1; C_1)}{MI_2(X_2; C_2)} = t \cdot \frac{\sum_{x_1, c_1} N_{x_1, c_1} \log_2 \frac{|S_1| N_{x_1, c_1}}{N_{x_1} N_{c_1}}}{\sum_{x_1, c_1} t \cdot N_{x_1, c_1} \log_2 \frac{|S_2| t \cdot N_{x_1, c_1}}{t^2 \cdot N_{x_1} N_{c_1}}}$$

Noting that  $|S_1| = |S_2|/t$ , the above simplifies to:

$$\frac{MI_1(X_1; C_1)}{MI_2(X_2; C_2)} = \frac{\sum_{x_1, c_1} N_{x_1, c_1} \log_2 \frac{|S_1| N_{x_1, c_1}}{N_{x_1} N_{c_1}}}{\sum_{x_1, c_1} N_{x_1, c_1} \log_2 \frac{|S_1| N_{x_1, c_1}}{N_{x_1} N_{c_1}}} = 1$$

---

<sup>3</sup> Suppose we have two tables A and B that contain rows with columns X and C and whose rows were generated by the same random vector modeled after some probability distribution. If we estimate the probability of occurrence of the pair (X,C) from A, and it so happens that the predicted frequency of occurrence of the pair is within the range [0,2] then we would expect the error to be high – because the “rounding error” – i.e. the fractional quantities in the predicted frequency can be quite high as a percentage of the integer floor of the predicted frequency. For example, suppose that table B is 100 times the length of A and that the pair (“heart attack”, true) is generated by a process where the expected occurrence of the latter pair is 1.5 times in the table A – but in actuality it only occurred 1 time. If we take the occurrence in A as an estimate of the frequentist probability, then we would expect the same pair to occur  $100 \cdot 1 = 100$  times in B – but based on the estimate of it occurring 1.5 times in A, we know the pair should actually occur 150 times on B (for an expected error of 50%). This sort of distortion would lessen if A was a longer table. For suppose that A and B were both 100 times longer. Then we would expect the pair to occur 150 times in A – with the frequentist probability converging to the true probability. By “sufficiently long” in the main text of this paper, we mean that the table should be sufficiently long so that the frequentist probabilities of occurrence of the pairs converge to “true” probability of the underlying process.

Thus under certain conditions (which are laid out in our assumptions), we have proven that mutual information values from  $T_1$  and  $T_2$  are comparable.

## References

- Chin, T. (2001). "Hacker gets access to Medical Records." American Medical News.
- Dalenius, T. (1986). "Finding a needle in a haystack – or identifying anonymous census records." Journal of Official Statistics 2(3): 329-336.
- Dempster, A., N. Laird, et al. (1977). "Maximum likelihood from incomplete data via the EM algorithm." Journal of the Royal Statistical Society Series B 39(1): 1–38.
- Felligi, I. and A. Sunter (1969). "A Theory for Record Linkage." Journal of the American Statistical Association 64(28).
- Frank Remson Field, III. (1985). Application of multi-attribute utility analysis to problems in materials selection. Department of Materials Science & Engineering. Cambridge, MA, Massachusetts Institute of Technology.
- Geman, S. and D. Geman (1984). "Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images." IEEE Transactions on Pattern Analysis and Machine Intelligence 6: 721-741.
- Gostin, L. (1997). "Health care information and the protection of personal privacy: ethical and legal considerations." Ann Intern Med 127(8 Pt 2): 683-90.
- Green-Armytage, J. (2006). "A Survey of Basic Voting Methods."
- Hines, M. (2006). "Enterprise Security Threats Increasingly Come from Within."
- Hodge, J. G., Jr., L. O. Gostin, et al. (1999). "Legal issues concerning electronic health information: privacy, quality, and liability." Jama 282(15): 1466-71.
- J.L. Schafer (1999). "Multiple imputation: A primer." Statistical Methods in Medical Research 8: 3-15.
- John Hagel, I. and M. Singer (1999). "Net Worth: Shaping Markets When Customers Make the Rules."
- Johnson, P. E. (2005). "Voting Systems."
- Mandl, K. D., P. Szolovits, et al. (2001). "Public standards and patients' control: how to keep electronic medical records accessible but private." Bmj 322(7281): 283-7.
- Mitchell, T. (1997). Machine Learning, McGraw Hill.

Ramoni, M. and P. Sebastiani (2001). "Robust Learning with Missing Data." Machine Learning 45(2): 147 - 170.

Schulze, M. (2003). "A New Monotonic and Clone-Independent Single-Winner Election Method." Voting Matters(17): 9-19.

Sweeney, L. (2002). "Comments of Latanya Sweeney, Ph.D., To the Department of Health and Human Services On Standards of Privacy of Individually Identifiable Health Information."

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

Sweeney, L. (2003). "Achieving k-anonymity privacy protection using generalization and suppression, ." International Journal on Uncertainty, Fuzziness and Knowledge-based Systems 10(5): 571-588.

United States Office of Health And Human Services, O. o. C. R. (2003). "HIPAA Privacy/Security/Enforcement Regulation Text."

Vinterbo, S. (2002). "Privacy: A machine learning view." Decision Systems Group/Harvard Medical School.

Walls, J. (2000). "Errant E-Mails Violate Privacy of Kaiser Members." San Francisco Chronicle.