# Computational Disclosure Control

## *A Primer on Data Privacy Protection*

by

Latanya Sweeney

# Table of Contents

# INDEX OF FIGURES

This work is dedicated to

**Carrie Sweeney**

Though her death took her from me while I was still quite young and my life has never been quite the same, she left me with three important gifts that have served me well during my lifetime: (1) the unconditional love she gave me has made it possible for me to love others; (2) my personal relationship with God, which she fostered, has carried me through many difficult times; and, (3) her belief in education made this work possible.

# Abstract

Today’s globally networked society places great demand on the dissemination and sharing of person-specific data for many new and exciting uses. Even situations where aggregate statistical information was once the reporting norm now rely heavily on the transfer of microscopically detailed transaction and encounter information. This happens at a time when more and more historically public information is also electronically available. When these data are linked together, they provide an electronic shadow of a person or organization that is as identifying and personal as a fingerprint even when the information contains no explicit identifiers, such as name and phone number. Other distinctive data, such as birth date and ZIP code, often combine uniquely and can be linked to publicly available information to re-identify individuals. Producing anonymous data that remains specific enough to be useful is often a very difficult task and practice today tends to either incorrectly believe confidentiality is maintained when it is not or produces data that are practically useless.

The goal of the work presented in this book is to explore computational techniques for releasing useful information in such a way that the identity of any individual or entity contained in data cannot be recognized while the data remain practically useful. I begin by demonstrating ways to learn information about entities from publicly available information. I then provide a formal framework for reasoning about disclosure control and the ability to infer the identities of entities contained within the data. I formally define and present *null-map*, *k-map* and *wrong-map* as models of protection. Each model provides protection by ensuring that released information maps to no, *k* or incorrect entities, respectively.

The book ends by examining four computational systems that attempt to maintain privacy while releasing electronic information. These systems are: (1) my Scrub System, which locates personally-identifying information in letters between doctors and notes written by clinicians; (2) my Datafly II System, which generalizes and suppresses values in field-structured data sets; (3) Statistics Netherlands' μ-Argus System, which is becoming a European standard for producing public-use data; and, (4) my *k*-Similar algorithm, which finds optimal solutions such that data are minimally distorted while still providing adequate protection. By introducing anonymity and quality metrics, I show that Datafly II can overprotect data, Scrub and μ-Argus can fail to provide adequate protection, but *k*-similar finds optimal results.

# Acknowledgments

I thank Professor Hal Abelson at MIT for his professional guidance, for his reading hundreds of pages that were excluded from this document, for his constantly setting deadlines and for his exhibiting patience as each deadline was missed or extended. He has been an good role model and guide to academic life. Thank you.

I also thank Peter Szolovits at MIT for providing an environment that made it possible for me to learn about and explore the early part of my work in this area independently, in the fulfillment of my own vision, with little financial concern and with no lack of quality time. A special thanks also goes to Daniel Weitzner for making time to review this material and for his patience during its creation.

Also at MIT, I thank Patrick Winston for being there over the years. In those occasions when I sought him out, he was always there, willing to give an ear or lend a hand and in those pivot points, he made the difference. Professor Sussman made me feel so connected to MIT and I thank him for his inquisitive spirit and lively conversations on any topic in any part of math, science or engineering. Also thanks to Patrick Thompson and Jon Doyle for reviewing and commenting on the earliest drafts of my work in this area. Also thanks to Norman Margolis and Tyrone Sealy for their general support and assistance. Finally at MIT, I thank the graduate students, staff and faculty of the Laboratory for Computer Science and of the Artificial Intelligence Lab for a fun and fascinating environment. In the Boston area, I thank the medical informatics community for sharing data with me so I could take a first look at issues regarding patient confidentiality, and along these lines, a special thanks goes to Isaac Kohane at Children's Hospital and to Octo Barnett at Massachusetts General Hospital.

The latter parts of this work were conducted at Carnegie Mellon University. I thank Mark Kamlet for his tremendous support and willingness to make my transition to the faculty a fantastic opportunity at every turn. Enormous gratitude goes to Debra Dennison for her administrative support. The H. John Heinz III School of Public Policy and Management has been an incredibly challenging and stimulating environment as so many disciplines come together under one roof and the willingness of the faculty to extend themselves is outstanding. Special thanks in general to Rema Padman, George Duncan, Marty Gaynor, Stephen Roehrig, Susan McElroy, Janet Cohen and Linda Babcock. I also thank Tom

Mitchell, Diane Stidle, Steve Fienberg and the members of the Center for Automated Learning in the School for Computer Science for a stimulating and fun environment.

In terms of the wider privacy community, I have to first give special thanks to Beverly Woodward. While we may not always agree on solutions or even characterizations of problems, her willingness to provide critical analysis and review and to take the time to read and comment on my work is greatly appreciated. It has strengthened my work and my thinking. Thank you Beverly.

In the privacy community, I have participated in more than 50 public debates, talks, presentations and investigations. I thank all who hosted these events and invited me to participate. These include Deanna Mool, Virginia deWolf, Laura Zayatz, and Bill Winkler. Among the organizations are the American Psychiatric Association, the medical societies of Maryland and of Massachusetts, the U.S. Bureau of the Census, the American Statistical Association, the American Medical Informatics Association, the U.S. Department of Health and Human Services, the National Library of Medicine, the National Research Council, the Centers for Disease Control Prevention, and the U.S. Senate, to name a few. Thanks to all those who shared a debate or a discussion with me, in public or private, and who hurled a criticism or a compliment, for you have all contributed in one way or another to this work. Special recognition is extended to A.G. Breitenstein, Denise Nagel, Robert Gellman, and Janlori Goldman. I also want to recognize David Korn, L.J. Melton, and Elliot Stone.

In conducting this work, I traveled extensively and shared ideas and thoughts with many researchers from many different areas. Thanks again to Bill Winkler at the U.S. Bureau of Census for introducing me to his work and that of other statisticians. Thanks to Hundepool at Statistics Netherlands for public conversations and for a copy of μ-Argus software. Thanks to all who hosted me and who shared a discussion or two, good or bad, down the right path or the wrong path, because they all helped me find my way. These include Gio Wiederhold at Stanford, members of the Database Group at Stanford, and members of the security group at SRI, including Pierangela Samarati, Steve Dawson, and Pat Lincoln.

I also thank those supporters, critics and discussion sharers who wish to remain anonymous and all those who would like to have been identified but whom I did not explicitly identify.

Let me also take this opportunity for a more personal note. I have been blessed to have to have in my life those who can see value in the roughest of ordinary looking rocks. They behold beauty where most of us see nothing of interest, and by their faith alone they transform stone into diamonds for us all to see. I gratefully acknowledge Chang Sook Barrett, Sylvia Barrett and Joseph Barrett for motivating and inspiring this work. Their commitment and belief in education continue to inspire and influence all who know them, and their unwavering belief in me will never be forgotten. I also thank Henry Leitner and Harvard University DCE for their continued support and commitment to providing educational opportunities to all. Finally, but not least, I humbly thank Joyce Johnson, Sylvia Barrett, Chang Barrett and Joseph Barrett for giving me love for a lifetime. This work has been supported in part by a Medical Informatics Training Grant (1 T15 LM07092) from the National Library of Medicine, a grant from the U.S. Bureau of the Census, and the H. John Heinz III School of Public Policy and Management at Carnegie Mellon University.

# Chapter 0 Preface

In this chapter, I describe the organization of this embodiment of work. This is done by:

(1) describing the work and my contributions;
(2) identifying the intended audiences; and then,
(3) outlining the overall organization of this book.

## 0.1 Description of work

In the following paragraphs I describe the work reported in this book by describing it in terms of its broader implications and promise as a line of research.

### 0.1.1 Computational disclosure control

The overall objective of the line of research encouraged by this work is to create architectural, algorithmic and technological foundations for the maintenance of the privacy of individuals, the confidentiality of organizations, and the protection of sensitive information, despite the requirement that information be released publicly or semi-publicly. Data holders are finding it increasingly difficult to produce anonymous and declassified information in today's globally networked society. Most data holders do not even realize the jeopardy at which they place financial, medical, or national security information when they erroneously rely on security practices of the past. Technology has eroded previous protections, leaving the information vulnerable. In the past, a person seeking to reconstruct private information was limited to visiting disparate file rooms and engaging in the labor-intensive review of printed material in geographically distributed locations. Today, one can access voluminous worldwide public information using a standard handheld computer and ubiquitous network resources. Thus, from seemingly innocuous anonymous data and available public and semi-public information, one can draw damaging inferences about sensitive information.

However, one cannot seriously propose that all information with any links to sensitive information be suppressed. Society has developed an insatiable appetite for all kinds of detailed information for many worthy purposes, and modern systems tend to distribute information widely. A goal of this work is to control the disclosure of data such that inferences about identities of people and organizations and about sensitive information contained in the released data cannot reliably be made. In this way, information that is practically useful can be shared freely with guarantees that it is sufficiently anonymous and declassified. I call this effort the study of *computational disclosure control*.

## Motivation for disclosure control

Computational disclosure control is inspired by the astonishing proliferation of public information made available on the Internet and recent access to inexpensive, fast computers with large storage capacities. These may now render many declassification standards ineffective. Shockingly, there remains a common incorrect belief that if data look anonymous, it is anonymous. Data holders will often remove all explicit identifiers, such as name, address, and phone number, from data so that other information contained in the data can be shared, incorrectly believing the identities of entities contained in the data cannot be inferred. Quite the contrary, de-identifying information provides no guarantee of anonymity. For example, released information often contains other data, such as birth data and ZIP code that in combination can be linked to publicly available information to re-identify individuals. As another example, when somewhat aged information is declassified differently by the Department of Defense than by the Department of Energy, the overall declassification effort suffers; by using two partial releases, the original may be reconstructed in its entirety.

## Promise of computational disclosure control

Because computational disclosure control can provide a responsible means for providing detailed medical data to researchers, financial information to economists, and military intelligence information to analysts, society can reap tremendous benefits in allocation of resources, financial efficiencies, and protection of national information interests. Of course, this is only possible because the abstracted data does not compromise individuals, organizations or national interests. Computational disclosure control provides the means to coordinate information from vast numbers of distributed data holders so that intended disclosure and declassification policies can be collectively enforced, even when related inferences may not have been explicitly stated. Determining optimal results requires new insight into measuring the usefulness of anonymous data and the effectiveness of the protection provided.

### 0.1.2 Contributions of this work

The major contributions to computer science stemming from this work include: (1) a formal framework for reasoning about disclosure control problems; (2) methods for integrating disclosure limitation techniques to achieve a given level of anonymity; (3) the introduction of formal protection models; and, (4) the definition of metrics to assess quality and anonymity. The major contributions to computer science and to public policy concern: (1) identifying the nature of disclosure control problems in today’s technological and legal settings; (2) demonstrating how today’s policies, practices and legislation do not provide adequate privacy protection; and (3) proposing directions for new policies that incorporate new disclosure control technology.

### 0.1.3 Learning information about entities

In more traditional computer science terms, this work can be characterized as one on learning – in particular, the learning of information about entities from data. Society is experiencing tremendous growth in the number and variety of data collected and shared about individuals, companies and other entities. When these seemingly innocuous facts are combined, strategic or sensitive knowledge can be learned about entities. *Data linkage* is the study of algorithms for learning information about entities from disparate pieces of entity-specific information. An example is linking information gathered on the World Wide Web with publicly available databases to reveal information about personal behaviors or relationships between people.

On the other hand, there is often an expectation of privacy (e.g., medical information) or a pledge of confidentiality (e.g., censuses and surveys) that accompanies shared data. *Disclosure control* is the study of algorithms for releasing information about entities such that the privacy of the individuals or other sensitive inferences that can be drawn from the data are controlled while the data remain practically useful.

There exists a symbiotic relationship between data linkage and disclosure control. Data linkage algorithms that exploit disclosure vulnerabilities in data identify ways in which disclosure control must improve. Conversely, if disclosure control is to provide data that are useful, such algorithms must identify the inferences that remain.

Over the past twenty-five years, pursuits in *record linkage* (a subset of data linkage that relies on the technique of probabilistic linking) and in disclosure control have utilized various statistical approaches. However, the nature and extent of data available today has led to a re-examination of these approaches as well as to the development of new computational methods, which are presented in this book.

## 0.2 Intended audiences

This book is intended for graduate students who want to learn to be data protectors in order to limit the knowledge others can gain from information that is publicly released. Conversely, students also learn to be data detectives in order to understand ways to gain strategic knowledge about individuals and other entities. It is assumed that the student reading this book has a working knowledge of computer programming, data structures and algorithms. In a class setting, students may be responsible for uncovering sensitive information about individuals by conducting experiments similar to those reported in Chapter 2. Then, students could assume the responsibility of producing public information for a data holder using privacy protection methods like those described in chapters 4 through 9. Students could then attempt to compromise each other’s released data and assess the anonymity of each release. Because of the sensitive nature of this work, it is imperative that students consider the related ethical and societal pressures inherent in this work. These issues are underscored in Chapter 2 and the broader challenges to society posed by the work are discussed further in the last chapter.

## Other audiences

Maintaining the privacy of individuals and the confidentiality of organizations which are contained in electronic information released for public or semi-public use affects a wide range of audiences whose concerns are as diverse as information warfare, financial credit, epidemiological research and data warehousing, to name a few. In addition there is growing public concern over privacy and confidentiality as they relate to information made available over the Internet. As a result, the systems and techniques discussed in this book are quite timely. Demand for information about my work has been constant and immediate and has stemmed from a wide range of audiences including national security efforts, the United States Bureau of the Census, the Massachusetts Department of Education, statistical offices, other government agencies, medical organizations and federal and state legislative committees working on medical privacy laws. Each of these contexts has brought additional richness to the work that extends beyond differences in vocabularies to also offer unique ways of looking at similar problems given different traditions and practical experiences.

### Releasing medical information

In this book, I present technical solutions in the context of real-world problems. For brevity, the bulk of the book concerns problems and solutions in releasing medical information even though some emphasis is placed on important distinctions necessary for other audiences such as those concerned with statistical, financial or marketing data. Producing anonymous medical information is often very difficult, as I show herein, and attempting to furnish such data provides fertile ground on which to explore the general nature of disclosure control problems and the effectiveness of proposed solutions. The tension between maintaining the privacy of the individual and sharing information for the benefit of society is more taut and more transparent with medical data than with other kind of person-specific data, which is why I use medical data as the primary example throughout.

## 0.3 How this work is organized

This book consists of three major parts. The first part, consisting of chapter 2, briefly reports on re-identification experiments I designed and conducted using publicly available information as a means of demonstrating the difficulties encountered when attempting to produce anonymous information in today’s technical setting. Simultaneously, this chapter shows how in today’s setting, publicly available information can be exploited to reveal sensitive information about individuals and so, it therefore serves as a reflection on explorations in data linkage techniques. The second part of this book, consisting of chapters 3 through 5, includes a formal framework I defined for reasoning about these kinds of problems and a formal presentation I devised that examines the use of common techniques to thwart unwanted data linkage efforts. In chapters 6 through 9, I present four computational systems, three of which I created and produced, that attempt to produce anonymous information for public use. Comparative results are provided to demonstrate the effectiveness of these systems in light of the re-identification experiments conducted in the first part. In the final part of this book, consisting of chapters 10, the problems and proposed computational solutions are briefly examined in terms of their potential impact on privacy legislation, practices and policies.

## 0.4 Computer technology used

Two different machines were used for the re-identification experiments reported in chapter 1, but much of the work could have been performed with only one machine and that machine need not have been as powerfully configured. However, these machines were available for the tasks. Each is described below.

Dell Inspirion 3200 laptop computer
Pentium II, 144MB RAM, 6GB hard drive, CDROM
External 1GB Jaz drive with SCSI PCMCIA adapter
Ethernet (and 56K modem) connection to Internet
Windows 98 operating system
Office 97 with Access

Dell Precision 610
Pentium II, 1GB RAM, 40GB hard drive, CDROM
Internal 1GB SCSI Jaz drive
Ethernet connection to Internet
Windows NT operating system
Office 97 with Access, Oracle, SQL Server

# Chapter 1 Non-Technical Overview

The purpose of this chapter is to provide a concise, non-technical overview of a new emerging area of study, which I term computational disclosure control. An objective of this document is to provide fundamental principles on which subsequent work in this area may build. It includes references to my work beyond what is actually covered in later chapters. This chapter is intended as an overview for the non-technical reader, who may not read some or all of the subsequent chapters. Other readers can skip this chapter with no loss of information.

Organizations often release and receive person-specific data with all explicit identifiers, such as name, address and telephone number, removed on the assumption that privacy is maintained because the resulting data look anonymous. However, in most of these cases, the remaining data can be used to re-identify individuals by linking or matching the data to other data bases or by looking at unique characteristics found in the fields and records of the data base itself. When these less apparent aspects are taken into account, each released record can be altered to map to many possible people, providing a level of anonymity that the record-holder determines. The greater the number of candidates per record, the more anonymous the data.

In this book, I present four general-purpose computer programs for maintaining privacy when disclosing person-specific information. They are:

- my Scrub System, which locates and suppresses or replaces personally identifying information in letters, notes and other textual documents;
- my Datafly II System, which generalizes values based on a profile of the data recipient at the time of disclosure;
- Statistics Netherlands' μ-Argus System, a somewhat similar system which is becoming a European standard for disclosing public use data; and,
- my *k*-Similar algorithm, which finds optimal results such that the data are minimally distorted yet adequately protected.

These systems have limitations. When they are completely effective, wholly anonymous data may not contain sufficient details for all uses. Care must be taken when released data can identify individuals and such care must be enforced by coherent policies and procedures that incorporate the constantly changing challenges posed by technology.

## 1.1 Towards all the data on all the people

There has been tremendous growth in the collection of information being collected on individuals and this growth is related to access to inexpensive computers with large storage capacities. Therefore, the trend in collecting increasing amounts of information is expected to continue. As a result, many details in the lives of people are being documented in databases somewhere and that there exist few operational barriers to restrict the sharing of collected information. In a related work, I proposed a formal mathematical model for characterizing real-world data sharing policies and defined privacy and risk metrics to compare policies. These metrics were applied to the real-world practices of sharing hospital discharge data. Findings include: (1) 25 of the 44 states that collect hospital discharge data share the information on a public or semi-public basis; (2) the number of people eligible to receive a copy of the data is greater than the number of people whose information is contained in the data; and, (3) publicly available data tends to be overly distorted and so more copies of the more sensitive, semi-publicly available data are more commonly distributed. Having so much sensitive information available makes it even more difficult for other organizations to release information that are effectively anonymous.

## 1.2 Unique and unusual values in statistical data

I conducted experiments using 1990 U.S. Census summary data to determine how many individuals within geographically situated populations had combinations of demographic values that occurred infrequently. It was found that combinations of few characteristics often combine in populations to uniquely or nearly uniquely identify some individuals. Clearly, data released containing such information about these individuals should not be considered anonymous. Yet, health and other person-specific data are publicly available in this form. Here are some surprising results using only three fields of information, even though typical data releases contain many more fields. It was found that 87% (216 million of 248 million) of the population in the United States had reported characteristics that likely made them unique based only on {*5-digit ZIP*, *gender*, *date of birth*}. About half of the U.S. population (132 million of 248 million or 53%) are likely to be uniquely identified by only {*place*, *gender*, *date of birth*}, where *place* is basically the city, town, or municipality in which the person resides. And even at the county level, {*county*, *gender*, *date of birth*} are likely to uniquely identify 18% of the U.S. population. In general, few characteristics are needed to uniquely identify a person.

### 1.3 Linking to re-identify de-identified data

I conducted experiments that demonstrated how de-identified health data can be linked to a population register in order to re-identify by name the persons who are the subjects of the health information. Using the voter list for Cambridge, Massachusetts, I showed how a few demographics combine to uniquely identify individuals. It was found that 12% of the 54,805 voters had unique birth dates (month, day and year of birth). Therefore, any information on these individuals that included birth date and city, would almost certainly be specific to the named individuals. Further, birth date and gender together were unique for 29%, birth date and a 5-digit ZIP (postal code) were unique for 69% and birth date and the full 9-digit ZIP were unique for 97% of the voters. These results demonstrate that combinations of characteristics can combine to construct a unique or near-unique identifier which is termed a *quasi-identifier*. These results further show that the typical de-identification technique applied when releasing information for public-use in the United States, does not render the result anonymous.

### 1.4 Probabilistic inference to re-identify individuals

I conducted an experiment in which five patients in a proposed release of cancer incidence information consisting of {*diagnosis*, *date of diagnosis* (month and year), *ZIP* (5 digits)} were accurately identified using only publicly available information. The method of re-identification concerned probabilistic inferences drawn from the Social Security Death Index based on population demographics and the specifics of the diseases. Four of the five cases had a diagnosis of Kaposi's Sarcoma which when found in young men is an indicator of AIDS. The fifth case concerned Neuroblastoma in a child and the re-identification was successful even though there is far less information available about children than about adults. It is difficult to believe that such seemingly minimal information could have been so easily re-identified.

### 1.5 Re-constructing unreleased data

I conducted an experiment in which a birth certificate database is reconstructed from publicly available information even though the state's vital records department did not release any of the information used. A total of 313 explicitly identified birth notices appeared in the Peoria Daily Record for April 1991. Hospital births found in publicly available health data reported 321 births for the same area during that time period which demonstrates a compliance of newspaper birth notices of 313/321 (or 98%). The combination of {*hospital*, *gender*, *date of birth*, *ZIP*/*place*} was unique for 234/313 (or 75%) of the births. The other 79 cases are described as follows. Twins (5 cases) and notices that could not be distinguished from one other notice (44 notices) were partitioned into 27 sets of two and accounted for 54 (or 17%) of the babies. In these cases, released information would be specific to one of the two named individuals. Similarly, there was one set of triplets and 18 other notices that could not be distinguished from two others; these were partitioned into 7 sets and accounted for 21 (or 7%) of the babies. Lastly, there were four notices that could not be distinguished on the basis of these attributes; these accounted for four (or 1%) of the notices. Additional sensitive inferences can be weakly implied from birth notices, such as the ethnicity of the child based on family name, family income based on residence, the child's general health at birth based on the timing of birth notices and the parent's marital status based on the absence of a father's name. Inferences from related hospital information can concern payment means or birth complications and anomalies, some of which may provide inferences to the mother's lifestyle or health. The resulting data can be used as a population register to re-identify individuals who later become the subjects of other releases of sensitive information.

## 1.6 Using patterns to re-identify individuals

I conducted a series of experiments that demonstrate how person-specific neuroblastoma incidence data, believed to be anonymous and being considered for release, could be re-identified using publicly available information. The proposed release consisted of 319 Illinois residents reported as being diagnosed with neuroblastoma from January 1986 through April 1998. Given only {*date of diagnosis* (month and year), *ZIP* (5-digit postal code in which each person resided)}, I employed linking and pattern matching techniques to re-identify these Illinois residents from seemingly innocent information. What is further surprising is that these experiments are among the most difficult possible because there is less publicly available information on children, who are the primary subjects, and because neuroblastoma is not a single, identified diagnosis code in health data. Instead, I showed that a series of diagnoses imply neuroblastoma. Information used for these experiments included Web pages, email discussion archives, health care data, Social Security death index, and birth notices. I correctly identified 20 of the 23 sampled (or 87%), uniquely identified 18 of the 23 sampled (or 78%) and incorrectly identified 0 of the 23 sampled.

## 1.7 Summary of problems producing anonymous data

Consider the re-identification experiments just described over the previous paragraphs. They reveal an array of problems encountered in attempting to produce anonymous information in today's technological setting. These problems center on:

(1) knowledge the recipient may hold or bring to bear on the data;
(2) unique and unusual combinations of values appearing within the data;
(3) an inability to prove a given release is anonymous.

Finding operational solutions to these problems is the topic of this work.

## 1.8 Related work

Prior related work comes from work in the statistics community on statistical databases and in the computer security community on multi-level databases, access control and authentication and inference control with respect to multiple queries to a database. While many techniques from these fields seek to effect disclosure control, they do so in different and more limited contexts than are explored in this work.

The reason for examining disclosure control in a broader context results from the dramatic increase in the availability of person-specific information from autonomous data holders. In the case of statistical databases, current demand centers on person-specific details and not aggregated summaries. In the case of multi-level databases, solutions can result from having absolute control over the entire collection and dissemination process. Such conditions are not possible with today's decentralized collections where release decisions are autonomously determined.

For the most part, computer security as a field has not addressed issues concerning data privacy that are separate and distinct from those of hardware security. Clearly, having competent hardware security can limit unwanted access to the information contained within the system, but having good security cannot guarantee privacy. As examples, consider the re-identification experiments mentioned earlier. In those cases, breaches of privacy resulted from data that were given out freely; no security breaches occurred.

## 1.9 Formal methods

There are numerous disclosure limitation techniques that can be brought to bear, but previously no formal protection models existed. I developed a formal framework for reasoning about disclosure control and the ability to infer the identities of entities contained within data. I also defined an *anonymous database system* as one that makes individual and entity-specific data available such that individuals and other entities contained in the released data cannot be reliably identified. I then introduced formal protection models, named *null-map*, *k-map* and *wrong-map*. Each model provides protection by ensuring that released information maps to no, *k* or incorrect entities, respectively.

Anonymous databases differ in many significant ways from statistical databases and from multi-level databases. Here are a few differences:

(1) all if not most of the data are released rather than a small sample;
(2) the integrity of entity-specific details must be maintained rather than an overall aggregate statistic; and,
(3) suppressing explicit identifiers, such as name and address, is not sufficient since combinations of other values, such as ZIP and birth date, can combine uniquely to re-identify entities.

My formal framework and protection models provide a basis for characterizing and comparing proposed anonymous database systems. Below are four real-world systems that are proposed to be anonymous database systems.

## 1.10 Scrub System

My Scrub System concerns maintaining privacy in textual documents. In field-structured databases, explicit identifiers, which provide a means to directly communicate with the person who is the subject of the data, appear within the data, grouped by a field name, such as {*name*, *phone number*}. Locating explicit identifiers in unrestricted text, however, becomes a problem unto itself. In the Scrub System, I define a new computational approach to locating and replacing personally identifying information in textual documents that extends beyond straight search-and-replace procedures, which was the previous norm. The system's approach is based on a model of how humans de-identify textual documents. The basic idea is to construct a system of detectors that work in parallel, where each detector specializes in recognizing a particular kind of explicit identifier.

While the Scrub System was proven to be quite effective, accurately locating 98-100% of all explicit identifiers found in letters to referring physicians, the final analysis reveals that de-identifying textual documents (i.e., removal of explicit identifiers) is not sufficient to ensure anonymity. Therefore, Scrub is not an anonymous database system. Nonetheless, de-identifying textual documents remains in great demand primarily due to archives of email messages, personal web pages and other information found on the World Wide Web and a lack of understanding of what renders data sufficiently anonymous.

### 1.11 Datafly II System

My Datafly II System concerns field-structured databases. Both my Datafly and Datafly II System use computational disclosure techniques to maintain anonymity in entity-specific data by automatically generalizing, substituting and removing information as appropriate without losing many of the details found within the data. For the discussion in this chapter, the terms Datafly and Datafly II can be consider to refer to the same basic system because the differences between them are not reflected in the issues presented here. Decisions are made at the attribute (field) and tuple (record) level at the time of database access, so the approach can be used on the fly in role-based security within an institution, and in batch mode for exporting data from an institution. As I mentioned in the experiments earlier, organizations often release person-specific data with all explicit identifiers, such as name, address, phone number, and social security number, removed in the incorrect belief that the identity of the individuals is protected because the resulting data look anonymous. However, the experiments showed that in most of these cases, the remaining data can be used to re-identify individuals by linking or matching the data to other databases or by looking at unique characteristics found in the attributes and tuples of the database itself. When these less apparent aspects are taken into account, as is done in my Datafly II System, each released tuple can be made to ambiguously map to many possible people, providing a level of anonymity that the data provider determines.

I term this model of protection *k*-map protection. In my Datafly and Datafly II System, the *k* is enforced on the data itself, resulting in a special form of *k*-map protection called *k*-anonymity. This is attractive because adherence to *k*-anonymity can be determined by the data holder's data alone and does not require omniscience. Further, in the Datafly System the data holder assigns to each attribute, the amount of tolerance for distortion that is desirable. Conversely, the provider of the data assigns to each attribute, the amount of protection necessary. In this way, the Datafly II System transforms the disclosure limitation problem into an optimization problem. As a consequence, the final results are adequately protected while remaining useful to the recipient. It is shown that Datafly is an anonymous database system.

## 1.12 μ-Argus System

The μ-Argus System is a computational disclosure system produced by Statistics Netherlands that is similar to my Datafly System. Both systems utilize the same disclosure limitation techniques to enforce *k*-anonymity and in both systems, the data provider assigns to each attribute, the amount of protection necessary though the granularity of this specification is far more coarse in μ-Argus. These similarities are especially surprising given that the systems were developed at roughly the same time and with no prior knowledge of each other; and, each work stems from a different academic tradition. But the systems differ in significant ways. In Datafly II each release is guaranteed to adhere to *k*-anonymity where such is not necessarily the case in μ-Argus. However, μ-Argus tends to provide less distortion than Datafly II so more of the specificity in the values themselves remains, making the data often more useful. It is shown that μ-Argus is not an anonymous database system.

## 1.13 The *k*-Similar algorithm

My *k*-Similar algorithm finds optimal solutions such that data are minimally distorted while still providing adequate protection. By introducing anonymity and quality metrics, I show to what extent Datafly II can over distort data, while Scrub and μ-Argus can fail to provide adequate protection in a given release. In contrast, my *k*-similar algorithm produces optimal releases that are not overly distorted nor under-protected. It does so by looking at the computational disclosure control problem as one of data clustering. In the well-known *k*-cluster algorithm, for example, data are partitioned into *k* groups based on minimizing a distance between tuples. In contrast, the *k*-similar algorithm divides data into groups such that the size of each group consists of *k* or more of the "closest" tuples; in this case, closeness is based on a minimal distance measure derived from the anonymity and quality metrics. In terms of computational speed, *k*-Similar operates in real-time under certain circumstances, but can become combinatoric in others. It is not nearly as fast as Datafly and μ-Argus. However, the resulting releases from *k*-Similar are guaranteed to be minimally distorted yet sufficiently protected which is not the case with the other systems.

## 1.14 Putting the systems into action

Revisiting the linkage experiments described in the earlier sections, given the computational solutions described in the later sections, shows that these solutions can effectively thwart the described re-identification efforts. Using the quality and anonymity metrics related to my formal methods, I conducted an experiment that demonstrated that public-use medical data available today is typically over-distorted yet still inadequately protected. This is not surprising given that these releases do not use any of the disclosure control systems presented here and do not employ any formal protection models. So, the impact of this work in the future should be significant.

## 1.15 Medical privacy legislation

While there may be many other possible academic approaches to protecting privacy, most of them are not practical in today's social settings. Therefore, it is important for those working in this area to understand the constraints the social setting places on the disclosure control problem. Consider medical privacy legislation, policies and best practices.

Policy makers appear to be unaware of the kinds of disclosure control problems examined herein and the role that technology plays in rendering our past approaches to privacy policies futile. Basically, no medical privacy legislation proposed by Congress addresses the problems demonstrated in the earlier sections. That is, if any were to pass, the problems would remain. Major shortcomings center on:

(1) an incorrect belief that de-identifying data renders the result anonymous;

(2) an incorrect belief that data linkage and re-identification can be controlled by encryption alone;

(3) an incorrect belief that following established computer security practices provides adequate privacy protection; and,

(4) an inability to construct a policy framework for privacy legislation that does not require enumerating all sources, recipients and uses of data a priori.

New technology offers better choices than the all-or-nothing positions voiced in the medical privacy debates, but technical solutions alone remain inadequate. Technology must work with policy for the most effective solutions.

## 1.16 Challenge to society

While medical data has been used to motivate the work described here, the problem is certainly not limited to medical data. Given the explosion in the collection and sharing of person-specific information described earlier, along with the growing ability to automatically process video and speech surveillance data and the ease of collecting information over the World Wide Web, populations are coming under increasingly intense data surveillance. For the United States, this is especially alarming because it undermines the philosophical cornerstones of the American way of life. It is not clear what terms like “freedom” and “liberty” mean in the absence of personal privacy. An inability to release entity-specific information that is anonymous is becoming one of the biggest and most significant challenges facing today’s society.

For example, the Freedom of Information Act has historically provided a mechanism to help ensure government accountability, but when many such releases are not effectively anonymous, they can easily become weapons to reveal sensitive information about individuals or businesses. Conversely, this becomes grounds on which the government refuses to release many of the kinds of information currently reported. Similarly, the American legal system requires law enforcement to acquire search warrants based on a review of evidence by a judge. However, by using the linkage techniques described earlier, law enforcement can gain access to sensitive information about members of the population without the protection of a search warrant or even a reported case. These are just two examples that show how an inability to provide entity-specific data that are not anonymous tears at the underpinnings of American society and begs for society to re-examine itself in the wake of these problems.

## 1.17 Summary

On the one hand, having so much information available about entities provides many new and interesting ways to conduct research, but on the other hand, having so much information available about entities makes it increasingly difficult to provide personal privacy. So, this book focuses on several of my contributions including a formal framework for reasoning about these kinds of problems, 3 computational solutions to tackle this problem and a set of anonymity and quality metrics to help characterize solutions. Despite these contributions, care must be taken to use policy to tie the technology that brought forth the problem with the technology that can offer solutions.

# Chapter 2 Introduction

Society is experiencing exponential growth in the number and variety of data collections as computer technology, network connectivity and disk storage space become increasingly affordable. Data holders, operating autonomously and with limited knowledge, are left with the difficulty of releasing information that does not compromise privacy, confidentiality or national interests. In many cases the survival of the database itself depends on the data holder's ability to produce anonymous data because not releasing such information at all may diminish the need for the data, while on the other hand, failing to provide proper protection within a release may create circumstances that harm the public or others. Ironically, the broad availability of public and semi-public information makes it increasingly difficult to provide data that are effectively anonymous.

Let me begin by introducing my terminology and explaining my use of medical privacy as a constant example. In general, I will discuss collections of information whose granularity of details are specific to an individual, a business, an organization or other entities and I term such collections, *entity-specific data*. If the entities represented in the data are individuals, then I may refer to the collection as *person-specific data*; however, even in these cases, the concepts being presented typically apply to broader collections of entity-specific data as well. By primarily using person-specific data and focusing on issues surrounding medical privacy, the motivations and risks often become transparent even though the underlying issues apply to many other kinds of data such as financial, statistical and national security information.

## 2.1 Tensions in releasing data

In the next two subsections, I look at different ways in which society has made decisions about sharing data, and I provide a way to reason about these findings. In the end, this examination motivates my use of medical data as an example throughout this work, even though the issues presented are not limited to medical data.

## Quality versus anonymity

There is a natural tension between the quality of data and the techniques that provide anonymity protection. Consider a continuum that characterizes possible data releases. At one end of the continuum are person-specific data that are fully identified. At the other end are anonymous data that are derived from the original person-specific data, but in which no person can be identified. Between these two endpoints is a finite partial ordering of data releases, where each release is derived from the original data but for which privacy protection is less than fully anonymous. See Figure 1.

The first realization is that any attempt to provide some anonymity protection, no matter how minimal, involves modifying the data and thereby distorting its contents. So, as shown in Figure 1, movement along the continuum from the fully identified data towards the anonymous data adds more privacy protection, but renders the resulting data less useful. That is, there exists some tasks for which the original data could be used, but those tasks are not possible with the released data because the data have been distorted.

So, the original fully identified data and the derived anonymous data are diametrically opposed. The entire continuum describes the domain of possible releases. Framed in this way, a goal of this work is to produce an optimal release of data so that for a given task, the data remain practically useful yet rendered minimally invasive to privacy.

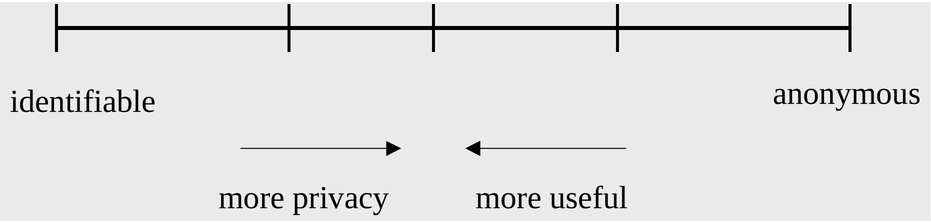
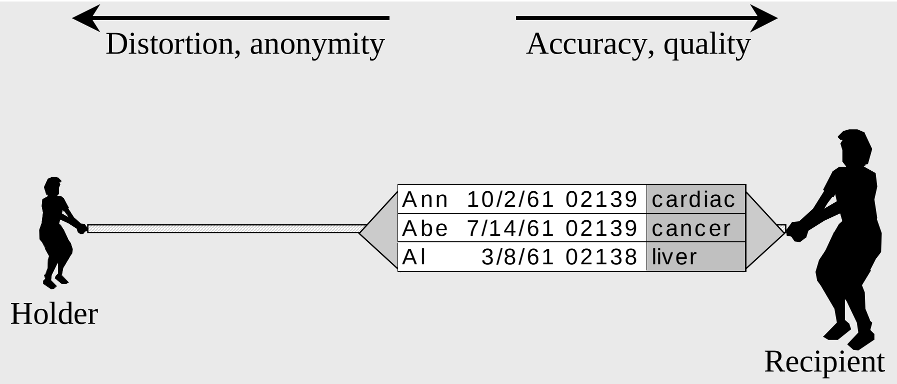


**Figure 1 Optimal releases of data**

## Tug-of-war between data holders and recipients

The second realization that emerges from Figure 1 is that the usefulness of data is determined by the task to which the recipient puts the data. That is, given a particular task, there exists a point on the continuum in Figure 1 that is as close to anonymous as possible, yet the data remain useful for the task. A release of data associated with that point on the continuum is considered optimal. In the next paragraphs, I provide a skeletal depiction of current practices that determine who gets access to what data. I show that the result can be characterized as a tug-of-war between data holders and data recipients.

In general, the practices of data holders and related policies do not examine tasks in a vacuum. Instead, the combination of task and recipient together are weighed against privacy concerns. This can be modeled as a tug-of-war between the data holder and societal expectations for privacy on one side, and the recipient and the recipient's use for the data on the other. In some cases such as public health legislation, the recipient's need for the data may overshadow privacy protections, allowing the recipient (a public health agent) to get the original, fully identified health data. See Figure 2 in which a tug-of-war is modeled. The privacy constraints on the data holder versus the recipient's demand for the data are graphically depicted by the sizes of the images shown. In the case illustrated, the recipient receives the original, fully identified data.



| | | | |
|---|---|---|---|
| Ann | 10/2/61 | 02139 | cardiac |
| Abe | 7/14/61 | 02139 | cancer |
| Al | 3/8/61 | 02138 | liver |

**Figure 2. Recipient's needs overpower privacy concerns**

Figure 3 demonstrates the opposite extreme outcome to that of Figure 2. In Figure 3, the data holder and the need to protect the confidentiality or privacy of the information overshadows the recipient and the recipient's use for the data and so the data is completely suppressed and not released at all. Data collected and associated with national security concerns provides an example. The recipient may be a news-reporting agent. Over time the data may eventually be declassified and a release that is deemed sufficiently anonymous provided to the press, but the original result is as shown in Figure 3, in which no data is released at all.

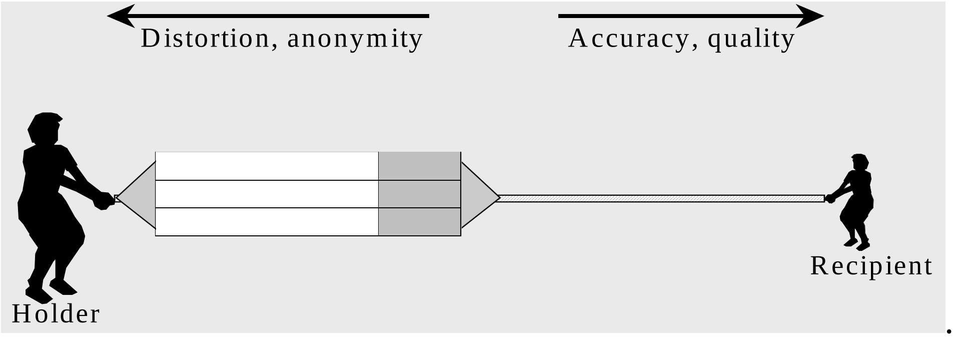


**Figure 3 Data holder and privacy concerns overpower outside uses of the data**

Figure 2 and Figure 3 depict situations in which society has made explicit decisions based on the needs of society as a whole. But secondary uses of medical data, for example, by marketing firms, pharmaceutical companies, epidemiological researchers and others do not in general lend themselves to such an explicit itemization. Figure 4 demonstrates situations in which the needs for privacy are weighed equally against the demand for the data itself. In such situations, a balance should be found in which the data are rendered sufficiently anonymous yet remain practically useful. As an example, this situation often occurs with requests by researchers for patient-specific medical records in which researchers seek to undertake clinical outcomes, or administrative research that could possibly provide benefits to society. At present, decisions are primarily based on the recipient receiving the original patient data or no data at all. Attempts to provide something in-between typically results in data with poor anonymity protection or data that is overly distorted. This work seeks to find ways for the recipient to get data that has adequate privacy protection, therefore striking an optimal balance between privacy protection and the data's fitness for a particular task.

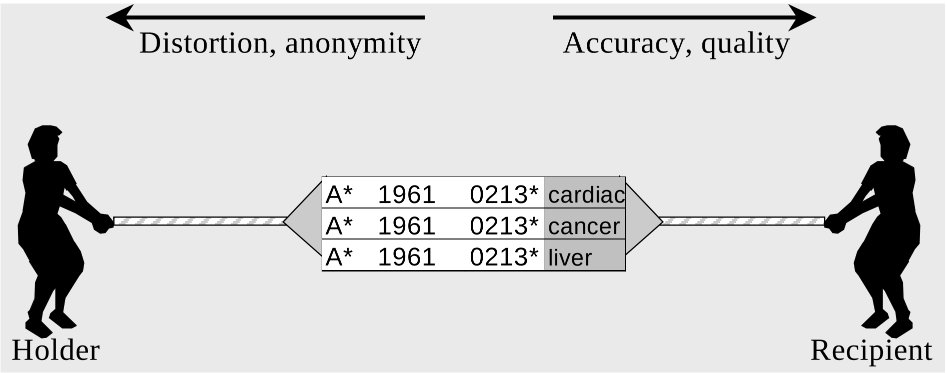


**Figure 4. An optimal balance is needed between privacy concerns and uses of the data**

At present, many data holders often make decisions arbitrarily or by ad hoc means. Figure 5 portrays the situation some state and federal agencies find themselves when they seek to produce public-use files for general use. Over the past few years, there has been a tremendous effort to make more data that is collected by government agencies available over the World Wide Web. In these situations, protecting the reputation of the agency, and the guarantees for privacy protection for which some agencies are legally bound, outweighs the demands of the recipient. In many of these cases, a strongly distorted version of the data is often released; the released data are typically produced with little or no consideration to the tasks required. Conversely, many other state and federal agencies release poorly protected data. In these cases, the individuals contained in the data can be easily re-identified. Examples of both of these kinds of released data are found in publicly and semi-publicly available hospital discharge data.

Neither way of releasing data yields optimal results. When strongly distorted data are released, many researchers cannot use the data, or have to seek special permission to get far more sensitive data than what are needed. This unnecessarily increases the volume of sensitive data available outside the agency. On the other hand, data that do not provide adequate anonymity may harm individuals.

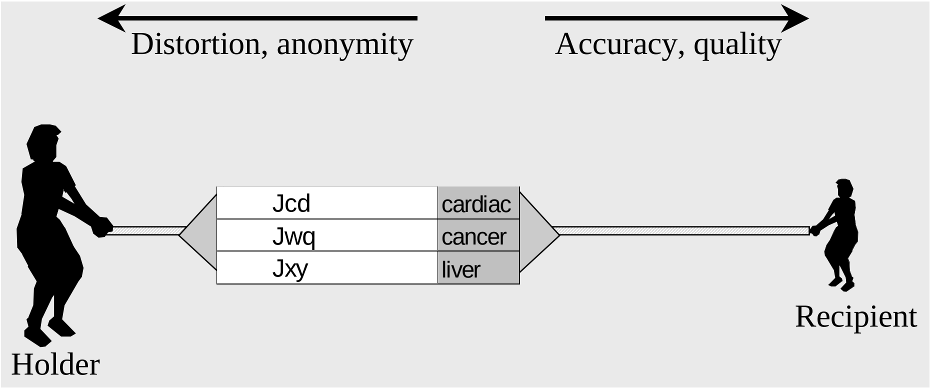


**Figure 5. Data holder and privacy concerns limit uses of the data**

In examining the different struggles between privacy and the sharing of person-specific data, I make the following claims:

Informal claim 1. Many current policies and practices support crude decisions. A recipient today too often receives the sensitive data itself, no data at all, overly distorted data that is of little or no use, or poorly protected data in which individuals can be re-identified.

Informal claim 2. Ultimately, the data holder must be held responsible for enforcing privacy protection because the data holder typically reaps a benefit and controls both data collection and dissemination.

While the claims above are independent of the content of data, the study of secondary uses of medical data in particular provides a natural incentive to find optimal solutions between researchers and data holders. After all, there are no legislative guidelines to empower one party so that it can overwhelm the other as was shown in Figure 2 and Figure 3. Also, state and federal agencies tend to be small in number and highly visible in comparison to the dramatic number of holders of medical data. Because there are so many holders of health data, it is hard to scrutinize their actions, and the resulting damage to individuals can be devastating yet hard to prove. And there exists strong financial incentives not to provide adequate protection in health data. On the other hand, research from data may lower health costs or save lives. For these reasons, focusing on the collection and sharing of medical data throughout this work provides motivation for finding optimal releases of data and for integrating technology with policy for maximal benefit. Even though I focus on anonymity protection in medical data, the issues presented are just as pertinent to the confidentiality of businesses, governments and other entities in financial, marketing and other forms of data.

## 2.2 Introduction to privacy in medical data

I begin with some informal definitions. *Identifiable personal health information* refers to any information concerning a person’s health or treatment in which the identity of the person can be determined. The expressions *personal health information* and *patient-specific health data* refer to health information that may or may not identify individuals. As I will show, in many releases of personal health information, individuals can be recognized. *Anonymous personal health information*, by contrast, contains details about a person’s medical condition or treatment but the identity of the person cannot be determined.

In general usage, confidentiality of personal information protects the interests of the organization while privacy protects the autonomy of the individual; but, in medical usage, both terms often mean privacy.

### 2.2.1 Privacy protection and the Hippocratic oath

The historical origin and ethical basis of medical confidentiality begin with the Hippocratic Oath, which was written between the sixth century BC and the first century AD:

"Whatsoever I shall see or hear in the course of my dealings with men, if it be what should not be published abroad, I will never divulge, holding such things to be holy secrets."

Various professional associations world-wide reiterate this oath, and by pledging this oath, clinicians – licensed professionals such as doctors, nurses, pharmacists, radiologists, and dentists who access in the line of duty identifiable personal health information – assume the responsibility of securing this information. The resulting trust is the cornerstone of the doctor-patient relationship, allowing patients to communicate with their physicians and to share information regarding their health status. However, the doctor-patient *privilege* offers very limited protection to patients regarding the confidentiality of their health information. Legal protection is very narrow, only applying in some cases when a physician is testifying in court or in related proceedings.

### 2.2.2 Role of information technology

The role of information technology is critical to confidentiality. On the one hand, information technology offers comprehensive, portable electronic records that can be easily accessed on behalf of a given patient no matter where or when a patient may need medical care [1]. That very portability, on the other hand, makes it much easier to transmit quickly and cheaply records containing identifiable personal health information widely and in bulk, for a variety of uses within and among health care institutions and other organizations and agencies. The Office of Technology Assessment (OTA) found that current laws generally do not provide consistent or comprehensive protection of personal health information [2]. Focusing on the impact of computer technology, OTA concluded that computerization reduces some concerns about privacy of personal health information while increasing others.

### 2.2.3 Past policy efforts and computational disclosure control

Previous policy efforts to protect the privacy of personal health information were limited to decisions about who gets access to which fields of information. I examine here four new computer programs that attempt to disclose information in such a way that individuals contained in the released data cannot be identified. These programs provide a spectrum of policy options. Decisions are no longer limited to who gets which fields of information, but to how much generality or possible anonymity will exist in the released information.

### 2.2.4 Public concern over privacy

The public's concern about the confidentiality of personal health information is reflected in a 1993 poll conducted by Harris and Associates for Equifax. The results of the survey found that 96 percent of the respondents believed federal legislation should designate all personal health information as sensitive, and should impose severe penalties for unauthorized disclosure. Eighty percent of respondents were worried about medical record privacy, and 25 percent had personal experience of abuse related to personal health information [3].

A 1994 Harris-Equifax consumer privacy survey focused on how the American public felt about having their medical records used for medical research and how safeguards would affect their opinions about such systems and uses. Among a list of thirteen groups and organizations, doctors and nurses ranked first in terms of the percentage of Americans who were "very" confident (43 percent) that this group properly handled personal and confidential information. After hearing a description about how medical records are used by researchers to study the causes of disease, 41 percent of Americans surveyed said they would find it at least somewhat acceptable if their records were used for such research without consent. Twenty-eight percent of those who initially opposed having their records used would change their position if a federal law made it illegal for any medical researcher to disclose the identity or any identifiable details of a person whose health records had been used. This would increase acceptance of this practice to over half those surveyed (58 percent) [4]. By extension, this survey implies strong public support for releases of personal health information in which persons contained in the information could not be identified.

### 2.2.5 Sharing medical data offers benefits to society

Analysis of the detailed information contained within electronic medical records promises many social advantages, including improvements in medical care, reduced institutional costs, the development of predictive and diagnostic support systems [5], and the integration of applicable data from multiple sources into a unified display for clinicians [6]. These benefits, however, require sharing the contents of medical records with secondary viewers such as researchers, economists, statisticians, administrators, consultants, and computer scientists, to name a few. The public would probably agree that these secondary parties should know some of the information in the record, but such disclosure should not risk identifying patients.

### 2.2.6 Lots of medical data available from many sources

Beverly Woodward makes a compelling argument that, to the public, patient confidentiality implies that only people directly involved in one’s health care will have access to one’s medical records, and that these health professionals will be bound by strict ethical and legal standards that prohibit further disclosure [7]. The public is not likely to accept the notion that records are “confidential” if large numbers of people have access to their contents.

In 1996, the National Association of Health Data Organizations (NAHDO) reported that 37 states had legislative mandates to electronically gather copies of personal health information from hospitals [8] for cost-analysis purposes. Community pharmacy chains, such as Revco, maintain electronic records for over 60 percent of the 2.4 billion outpatient prescriptions dispensed annually. Insurance claims typically include diagnosis, procedure and medication codes along with the name, address, birth date, and SSN of each patient. Pharmaceutical companies run longitudinal studies on identified patients and providers. As more health maintenance organizations and hospitals merge, the number of people with authorized access to identifiable personal health information will increase dramatically because, as the National Research Council (NRC) recently warned, many of these systems allow full access to all records by any authorized person [9]. For example, assume a billing clerk at hospital X can view all information in all medical records within the institution. When hospital X merges with hospitals Y and Z, that same clerk may then be able to view all records at all three hospitals, even though the clerk may not need to know information about the patients at the other institutions.

### 2.2.7 Problems have been found

The NRC report also warns against inconsistent practices concerning releases of personal health information. If I approach a hospital as a researcher, I must petition the hospital's institutional review board (IRB) and state my intentions and methodologies; then the IRB decides whether I get data and in what form. But, if I approach the same hospital as an administrative consultant, data are given to me without IRB review. The decision is made and acted on locally.

Recent presentations by the secretary of the Department of Health and Human Services emphasize the threats to privacy stemming from misuse of personal health information [10]. There have been abuses; here are just a few:

- A banker reportedly cross-referenced a list of patients with cancer against a list of people who had outstanding loans at his bank. Where he found matches, he called in the outstanding loans [11].
- A survey of 87 Fortune 500 companies with a total of 3.2 million employees found that 35 percent of respondents used medical records to make decisions about employees [12].
- Cases have been reported of snooping in large hospital computer networks by hospital employees [13], even though the use of a simple audit trail – a list of each person who looked up a patient's record – could curtail such behavior [14].
- *Consumer Reports* found that 40 percent of insurers disclose personal health information to lenders, employers, or marketers without customer permission [15].

Abuses like the preceding underscore the need to develop safeguards.

## 2.3 All the data on all the people

Before I look at inference problems inherent in producing anonymous information, I first want to consider why concern over the problem appears to be escalating. There is currently unprecedented growth in the number and variety of person-specific data collections and in the sharing of this information. The impetus for this explosion has been the proliferation of inexpensive fast computers with large storage capacities operating in ubiquitous network environments.

In an attempt to characterize the growth in person-specific data, I introduce a new metric termed global disk storage per person or GDSP, which is measured in megabytes per person. GDSP is the total rigid disk drive space in megabytes of new units sold in a year divided by the world population in that year. Figure 6 uses GDSP figures to compute the amount of a person's time that can be documented on a page of text using a regularly spaced fixed font.

| | **1983** | **1996** | **2000** |
|---|---|---|---|
| **Storage space** (TB) | 90 | 160,623 | 2,829,288 |
| **Population** (million) | 4,500 | 5,767 | 6,000 |
| **GDSP** (MB/person) | 0.02 | 28 | 472 |
| **Time per page** | 2 months | 1 hour | 3.5 minutes |

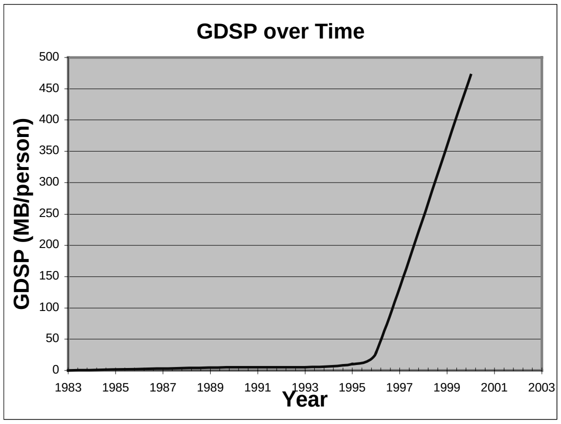


**Figure 6 Global disk storage per person**

In 1983 a half a page could be used to document each month of a person's life in that year. These recordings included itemized long distance phone calls, credit card purchases, volume of electricity used, and so forth. In 1996, a page could be used to document each hour of a person's life. Recordings expanded in both size and number. Examples of new collections included items purchased at the grocery store, web sites visited, and the date and time in some locations a car proceeded through a tollbooth. By the year 2000, with 20 gigabyte drives leading the industry, it is projected that a page could be used to document every 3.5 minutes of a person's life. Most likely collections will expand to include biometric information such as, heart rate, pulse and temperature. One of the leading proponents of the information explosion is the health care industry, acting in the belief that having such information will help reduce cost and improve care.

| Examples | 1983 | 1996 |
|---|---|---|
| Each birth | 280 | 1864 |
| Each hospital visit | 0 | 663 |
| Each grocery visit | 32 | 1272 |

**Figure 7 Estimated growth in data collections (per encounter) in Illinois (in bytes)**

Figure 7 demonstrates how some data collections expanded from 1983 to 1996 for some person-specific encounters in the State of Illinois. The values are the number of *bytes* (letters, digits and other printable characters) that were stored for each person per encounter in the collection shown.

These examples exemplify recent behavioral tendencies recently found in the collection practices of person-specific data. These informally observed "trends" are enumerated below.

**Behavior 1.** Given an existing person-specific data collection, expand the number of fields being collected. I casually refer to this as the "*collect more*" trend.

**Behavior 2**. Replace an existing aggregate data collection with a person-specific one. I casually refer to this as the "*collect specifically*" trend.

**Behavior 3.** Given a question or problem to solve or merely provided the opportunity, gather information by starting a new person-specific data collection related to the question, problem or opportunity. I causally refer to this as the "*collect it if you can*" trend.

No matter how you look at it, all three tendencies result in more and more information being collected on individuals. Not only has there been a dramatic increase in the collection of person-specific data, but also in the sharing of collected data. I define four classes of access restrictions to person-specific data based on current practices. These are described in Figure 8.

**Insiders only (Pr) “*private*”.**
Data collections that are available to authorized “insiders only” are considered to be privately held information because the only people who gain access are almost exclusively those who directly collected the information.

**Limited Access (SPr) “*semi-private*”.**
Data collections denoted as having “limited access” are those where access extends beyond those who originally collected the information, but only an identifiable small number of people are eligible for access in comparison to a substantially larger number of people who are not eligible for access. This access policy typically includes an extensive application and review process.

**Deniable Access (SPu) “*semi-public*”.**
Data collections having “deniable access” are those where an application and review process may exist but only an identifiable small number of people are denied access in comparison to a substantially larger number of people who are eligible for access.

**No restrictions (Pu) “*public*”.**
Data collections having “no restrictions” are those where an application process may or may not exist, but the data collections are generally made available to all who request them.

**Figure 8 Levels of access restrictions by data holders to person-specific data**

There is no doubt that society is moving towards an environment in which society could have almost all the data on all the people. As a result, data holders are increasingly finding it difficult to produce anonymous and declassified information in today’s globally networked society. Most data holders do not even realize the jeopardy at which they place financial, medical, or national security information when they erroneously rely on security practices of the past. Technology has eroded previous protections leaving the information vulnerable. In the past, a person seeking to reconstruct private information was limited to visiting disparate file rooms and engaging in labor-intensive review of printed material in geographically distributed locations. Today, one can access voluminous worldwide public information using a standard handheld computer and ubiquitous network resources. Thus from seemingly anonymous data, and available public and semi-public information, one can often draw damaging inferences about sensitive information. However, one cannot seriously propose that all information with any links to sensitive information be suppressed. Society has developed an insatiable appetite for all kinds of detailed information for many worthy purposes, and modern systems tend to distribute information widely.

Primarily society is unaware of the loss of privacy and its resulting ramifications that stem from having so much person-specific information available. When this information is linked together it can provide an image of a person that can be as identifying as a fingerprint even if all explicit identifiers like name, address, and phone number are removed. Clearly a loss of dignity, financial income and credit worthiness can result when medical information is widely and publicly distributed. A goal of the work presented in this book is to control the release of data such that inferences about the identities of people and organizations and other sensitive information contained in the released data cannot be reliably made. In this way, information that is practically useful can be shared with guarantees that it is sufficiently anonymous and declassified. I call this effort the study of *computational disclosure control.*

In the next section, I introduce the basic problems of producing anonymous data.

## 2.4 Problems producing anonymous data

I now present examples that demonstrate why the problem of producing anonymous data is so difficult. Consider the informal definition of anonymous data below. While it is easy to understand what anonymous data mean, I will show by examples that it is increasingly difficult to produce data that are anonymous.

### Definition (informal). anonymous data

> The term anonymous data implies that the data cannot be manipulated or linked to identify an individual.

A common incorrect belief is that removing all explicit identifiers from the data will render it anonymous; see the informal definition of de-identified data below. Many policies, regulations and legislation in the United States equate de-identified data and anonymous data.

### Definition (informal). de-identified data

> De-identified data result when all explicit identifiers such as name, address, and phone number are removed, generalized, or replaced with a made up alternative.

Data holders often collect person-specific data and then release derivatives of collected data on a public or semi-public basis after removing all explicit identifiers, such as name, address and phone number. Evidence is provided in this chapter that this process is not sufficient to render data anonymous because combinations of attributes often combine uniquely to re-identify individuals.

### 2.4.1 A single attribute

The frequency with which a single characteristic occurs in a population can help identify individuals based on unusual or outlying information. Figure 9 contains a frequency distribution of birth years found in the list of registered voters for Cambridge, Massachusetts as of February 1997 [16]. It is not surprising to see fewer people present with earlier birth years. Clearly, a person born in 1900 in Cambridge is unusual and by implication less anonymous in data.

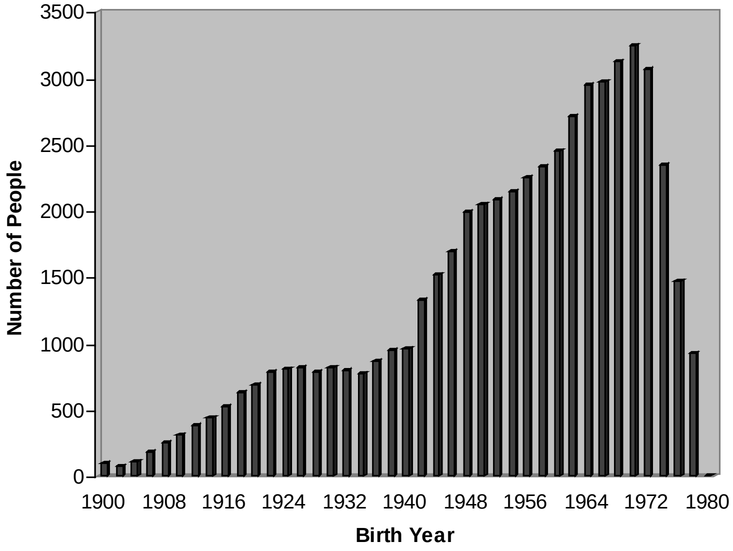
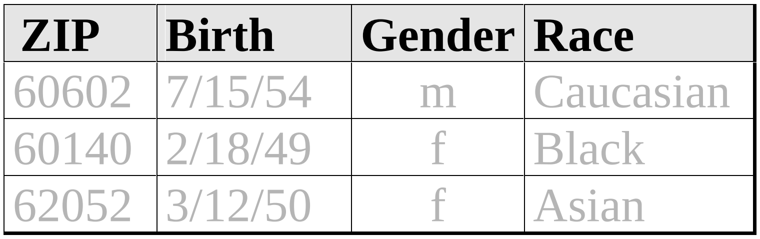


**Figure 9 Frequency of birth years in Cambridge Voter List**

### 2.4.2 More than one attribute

What may be more surprising is that combinations of characteristics can combine to occur even less frequently than the characteristics appear alone.

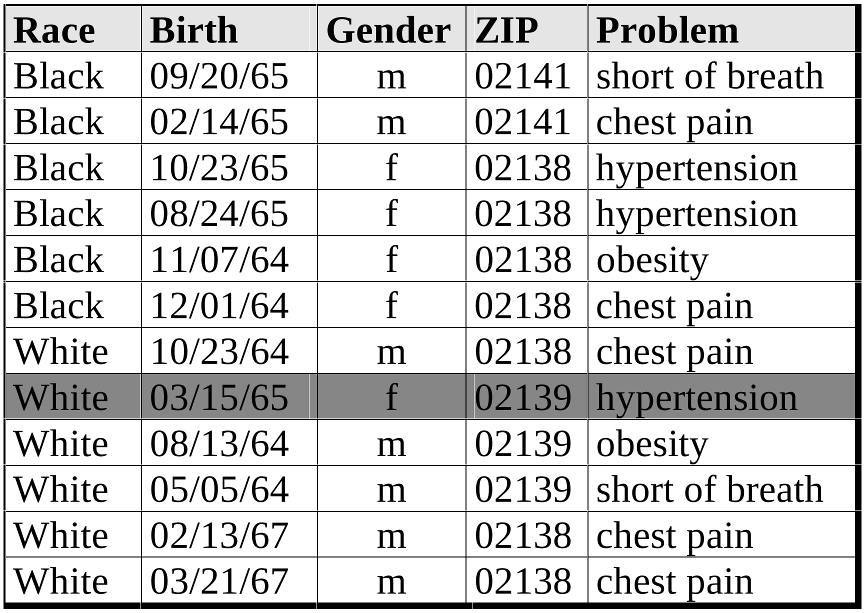


| ZIP | Birth | Gender | Race |
|---|---|---|---|
| 60602 | 7/15/54 | m | Caucasian |
| 60140 | 2/18/49 | f | Black |
| 62052 | 3/12/50 | f | Asian |

**Figure 10 Data that look anonymous**

Consider Figure 10. If the three records shown were part of a large and diverse database of information about Illinois residents, then it may appear reasonable to assume that these three records would be anonymous. However, the 1990 federal census [17] reports that the ZIP (postal code) 60602 consisted primarily of a retirement community in the Near West Side of Chicago and therefore, there were very few people (less than 12) of an age under 65 living there. The ZIP code 60140 is the postal code for Hampshire, Illinois in Dekalb county and reportedly there were only two black women who resided in that town. Likewise, 62052 had only four Asian families and the census further revealed that each of these households were headed by Filipino women and all their children were under 18 years of age. In each of these cases, the uniqueness of the combinations of characteristics found could help re-identify these individuals.

As another example, Figure 11 contains de-identified data. Each row contains information on a distinct person, so information about 12 people is reported. The table contains the following fields of information {*Race/Ethnicity*, *Date of Birth*, *Gender*, *ZIP*, *Medical Problem*}.



| Race | Birth | Gender | ZIP | Problem |
|---|---|---|---|---|
| Black | 09/20/65 | m | 02141 | short of breath |
| Black | 02/14/65 | m | 02141 | chest pain |
| Black | 10/23/65 | f | 02138 | hypertension |
| Black | 08/24/65 | f | 02138 | hypertension |
| Black | 11/07/64 | f | 02138 | obesity |
| Black | 12/01/64 | f | 02138 | chest pain |
| White | 10/23/64 | m | 02138 | chest pain |
| White | 03/15/65 | f | 02139 | hypertension |
| White | 08/13/64 | m | 02139 | obesity |
| White | 05/05/64 | m | 02139 | short of breath |
| White | 02/13/67 | m | 02138 | chest pain |
| White | 03/21/67 | m | 02138 | chest pain |

**Figure 11 De-identified data**

In Figure 11, there is information about an equal number of African Americans (listed as *Black*) as there are Caucasian Americans (listed as *White*) and an equal number of men (listed as *m*) as there are women (listed as *f*), but in combination, there appears only one Caucasian female. No Asian Americans are listed in Figure 11. These distributions are shown in Figure 12.

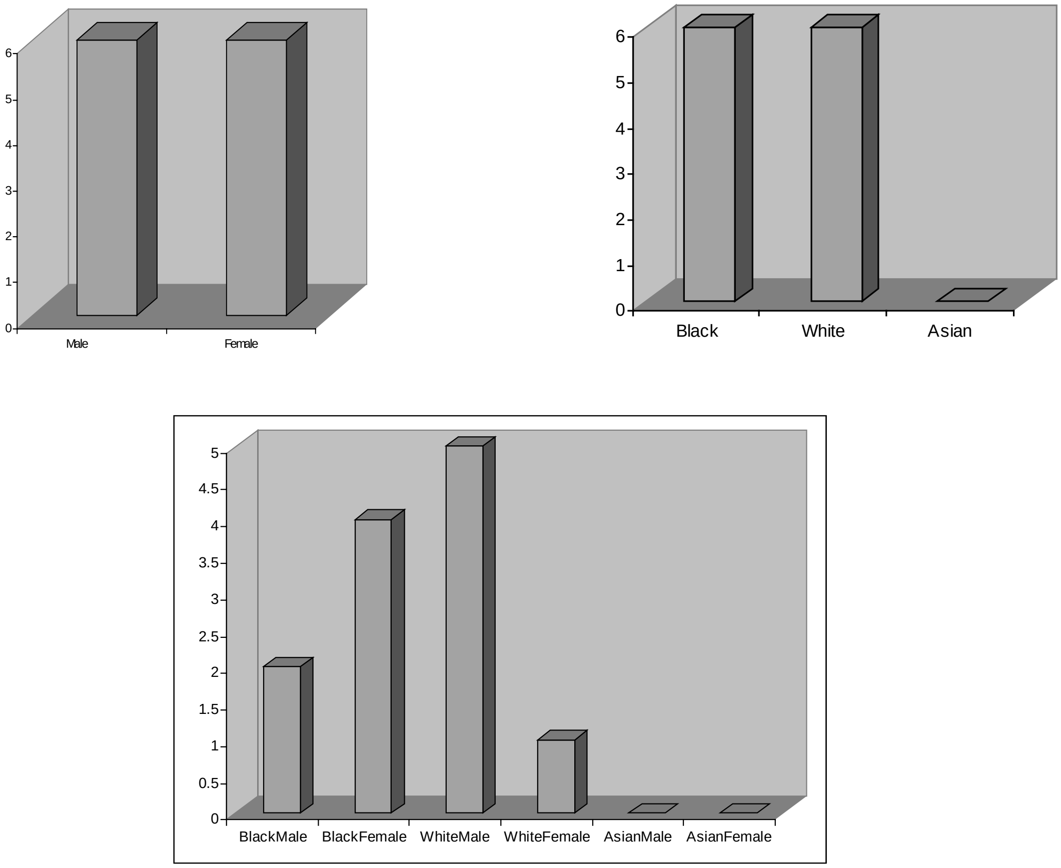


**Figure 12 Distributions of gender and race in Figure 11**

### 2.4.3 Learned from the examples

These examples demonstrate that in general, the frequency distributions of combinations of characteristics have to be examined in combination with respect to the entire population in order to determine unusual values and cannot be generally predicted from the distributions of the characteristics individually. Of course, obvious predictions can be made from extreme distributions --such as values that do not appear in the data will not appear in combination either. As an example, there were no Asians listed in Figure 11 and so, there were no Asian females or Asian males listed either.

### 2.4.4 Real-world examples

| Diagnosis | Diagnosis date | ZIP |
|---|---|---|
| … | … | … |
| … | … | … |
| … | … | … |
| … | … | … |
| … | … | … |

**Figure 13 Cancer registry that looks anonymous**

Recently, a state Department of Public Health received a Freedom of Information request from a newspaper that was researching occurrences of a rare cancer in a small region of the state. Although the paper only wanted diagnosis, date of diagnosis (month, day and year) and ZIP code (5 digits) for each patient in question, the state refused claiming that sensitive information might be gleamed from these data. In an attempt to discover how anonymous such information in question could be, I conducted an experiment. Within a few hours the name, and in some cases the Social Security number of five out of five patients submitted were accurately identified using only publicly available information. Further, four of the five cases had a diagnosis of Kaposi's Sarcoma which when found in young men is an indicator of AIDS and revealing such may have been prohibited by state law. Figure 13 shows an example of this data schema. A more extensive re-identification experiment, using similar data and achieving similar results was performed on cancer data with respect to children. It is difficult to believe that such seemingly innocuous information can be so easily re-identified.

- Patient **ZIP Code**
- Patient **Birth Date**
- Patient **Gender**
- Patient Racial Background
- Patient Number
- Visit Date
- Principal Diagnosis Code (ICD9)
- Procedure Codes (up to 14)
- Physician ID#
- Physician ZIP code
- Total Charges

**Figure 14 Attributes often collected statewide**

I will now demonstrate how linking can be used to perform such re-identifications. The National Association of Health Data Organizations (NAHDO) reported that 37 states have legislative mandates to collect hospital level data and that 17 states have started collecting ambulatory care data from hospitals, physicians offices, clinics, and so forth [18]. Figure 14 contains a subset of the fields of information, or attributes, that NAHDO recommends these states accumulate. The few attributes listed in Figure 14 include the patient’s ZIP code, birth date, gender, and ethnicity. Clearly, the data are de-identified. The patient number in earlier versions was often the patient's Social Security number and in subsequent versions was a scrambled Social Security number [19]. By scrambled I mean that the digits that compose the Social Security number are moved around into different locations. If a patient’s record is identified and their Social Security number known, then the scrambling algorithm can be determined and used to identify the proper Social Security numbers for the entire data set.

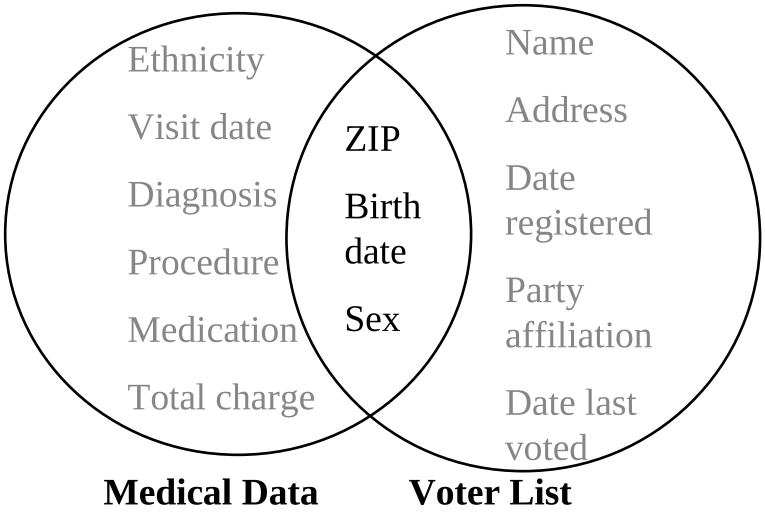


**Figure 15 Linking to re-identify data**

For twenty dollars I purchased the voter registration list for Cambridge Massachusetts and received the information on two diskettes [20] in an attempt to complete the re-identification. Figure 15 shows that these data included the name, address, ZIP code, birth date, and gender of each voter. This information can be linked using ZIP code, birth date and gender to the medical information described in Figure 14, thereby linking diagnosis, procedures, and medications to particularly named individuals. The question that remains of course is how unique would such linking be.

The 1997 voting list for Cambridge Massachusetts contained demographics on 54,805 voters. Of these, birth date, which is the month, day, and year of birth, alone could uniquely identify the name and address of 12% of the voters. One could identify 29% of the list by just birth date and gender; 69% with only a birth date and a five-digit zip code; and 97% when the full postal code and birth date were used.

Notice that these are only one and two way combinations and do not include three way combinations or beyond. These values are summarized in Figure 16.

| Attribute Combinations | Uniqueness |
|---|---|
| Birth date alone (mm/dd/yr) | 12% |
| Birth date and gender | 29% |
| Birth date and 5-digit ZIP | 69% |
| Birth date and full postal code | 97% |

**Figure 16 Value uniqueness in voter list**

In general I can say that the greater the number and detail of attributes reported about an entity, the more likely that those attributes combine uniquely to identify the entity. For example, in the voter list, there were 2 possible values for gender and 5 possible five-digit ZIP codes; birth dates were within a range of 365 days for 100 years. This gives 365,000 unique values, but there were only 54,805 voters.

I conducted experiments using 1990 U.S. Census summary data to determine how many individuals within geographically situated populations had combinations of demographic values that occurred infrequently. It was found that 87% (216 million of 248 million) of the population in the United States had reported characteristics that likely made them unique based only on {*5-digit ZIP*, *gender*, *date of birth*}. About half of the U.S. population (132 million of 248 million or 53%) are likely to be uniquely identified by only {*place*, *gender*, *date of birth*}, where *place* is basically the city, town, or municipality in which the person resides. And even at the county level, {*county*, *gender*, *date of birth*} are likely to uniquely identify 18% of the U.S. population. In general, few characteristics are needed to uniquely identify a person.

In Massachusetts, the Group Insurance Commission (GIC) is responsible for purchasing health insurance for state employees. GIC collected de-identified patient-specific data with nearly one hundred fields of information per encounter along the lines of the fields discussed in the NAHDO list for approximately 135,000 state employees and their families. Because the data were believed to be anonymous, GIC gave a copy of the data to researchers and sold a copy to industry [21]. William Weld was governor of Massachusetts at that time and his medical records were in that data. Governor Weld lives in Cambridge Massachusetts. According to the Cambridge Voter list, six people had his particular birth date; only three of them were men; and, he was the only one in his five-digit zip code.

Clearly the risks of re-identifying data depend both on the content of released data and on other related information. Most municipalities and states sell population registers such as voter lists, local census data, birth records and motor vehicle information. There are other sources of population registers such as trade and professional association lists. Such information can often be uniquely linked to de-identified data to provide names, addresses, and other personal information.

These real-world examples demonstrate two major difficulties in providing anonymous data: (1) knowledge a viewer of the data may hold or bring to bear on the data is usually not known beforehand by the data holder at the time of release; and, (2) unique and unusual values and combinations of values appearing within the data themselves often makes identification of related entities easier. The examples also underscore the need to develop solutions that limit the ability to link external information to data and therefore control the inferences that can be drawn.

The outline for the remainder of this work is as follows. In the next chapter, chapter 3, I discuss related work. I then survey disclosure control techniques and the nature of disclosure control in chapter 4. A formal presentation with accompanying definitions of protection models is also presented in chapter 4. Finally, four systems are presented and compared in chapter 5.

# Chapter 3 Background

The problem of controlling inferences that can be drawn from released data is not new. There are existing works in the statistics community on statistical databases and in the computer security community on multi-level databases to consider. However, none of these works provide solutions to the broader problems experienced in today's setting that are the topic of this work. Before examining these traditions, I establish a common vocabulary by adopting the following definitions.

Unless otherwise stated, the term *data* refers to entity-specific information that is conceptually organized as a table of rows (or records) and columns (or fields). Each row is termed a *tuple*. A tuple contains a relationship among the records or set of values associated with an entity. Tuples within a table are not necessarily unique. Each column is called an *attribute* and denotes a field or semantic category of information that is a set of possible values; therefore, an attribute is also a domain. Attributes within a table are unique. So by observing a table, each row is an ordered *n*-tuple of values $<d_1, d_2, \ldots, d_n>$ such that each value $d_j$ is in the domain of the *j*-th column, for $j$=1, 2, …, *n* where *n* is the number of columns. In mathematical set theory, a relation corresponds with this tabular presentation, the only difference is the absence of column names. Ullman provides a detailed discussion of relational database concepts [22].

Throughout the remainder of this work each tuple is assumed to be specific to one entity and no two tuples pertain to the same entity. This assumption simplifies discussion without loss of applicability.

To draw an *inference* is to come to believe a new fact on the basis of other information. A *disclosure* means that explicit or inferable information about an entity was released that was not intended. This definition may not be consistent with colloquial use but is used in this work consistent with its meaning in statistical disclosure control. So, *disclosure control* attempts to identify and limit disclosures in released data. Typically the goal of disclosure control with respect to person-specific data is to ensure that released data are anonymous.

## 3.1 Statistical databases

Federal and state statistics offices around the world have traditionally been entrusted with the release of statistical information about all aspects of the populace [23]. The techniques, practices and theories from this community however, have historically had three tremendous advantages. First, most statistics offices held centralized, sole-source exhaustive collections of information and therefore could often determine the sensitivity of many values using their data alone. Second, statistics offices primarily produced summary data, which by the nature of aggregation could often hide entity-specific information though care still had to be taken to protect against inferences. Finally, statistics offices previously released information in an environment whose computational power and access to other data was extremely limited. These advantages have been eroded in today's environment. Today's producers of useful publicly available data must contend with autonomous releases of entity-specific information by other data holders and with recipients who are technologically empowered.

Like other data holders, statistics offices are also facing tremendous demand for entity-specific data for applications such as data mining, cost analysis, fraud detection and retrospective research. But many of the established statistical database techniques, which involve various ways of adding noise [24] to the data while still maintaining some statistical invariant [25, 26], often destroy the integrity of tuples and so, for many new uses of data, these established techniques are not appropriate. I will further discuss disclosure limitation techniques commonly employed to protect the confidentiality of statistical databases in chapter 4; Willenborg and De Waal [27] provide more extensive coverage. However, I will mention Markov perturbation now as an example of a technique used in statistical disclosure control [28].

Given local census data that includes income, number of children and age, values can be slightly perturbed so overall statistics remain the same, but specific values are no longer available, thereby making it harder to link the information to other sources with confidence. Examples of such actions include: (1) decrementing the value associated with the *child* attribute in one tuple and then incrementing the value associated with a *child* attribute in another; and, (2) reducing the value associated with a *salary* attribute by $10,000 in one tuple and then adding $5000 to the values of two others. Unfortunately, many new applications that learn from data and detect correlation rely on the integrity of the tuple. Also many statistical disclosure limitation techniques have severely limited applicability because many new data collections are characterized as having primarily categorical attributes and not continuous ones. In a medical database, for example, how does one perturb a diagnosis of lung cancer?

Summary data is the result of aggregating information. Even in releases of summary data statistical offices are finding their established practices failing given the increase of entity-specific data and the proliferation of computing power because more data and more powerful tools are available for unwanted linking. The European Union in response to these growing concerns has recently funded a tremendous effort to develop solutions. Their first computational result was μ-Argus from Statistics Netherlands [29]. I will examine this system in chapter 5 and show the first release of μ-Argus does not provide adequate protection.

## 3.2 Multi-level databases

Another related area is aggregation and inference in multi-level databases [30, 31, 32, 33, 34, 35] which concerns restricting the release of lower classified information such that higher classified information cannot be derived. Denning and Lunt [36] described a multilevel relational database system (MDB) as having data stored at different security classifications and users having different security clearances.

Su and Ozsoyoglu [37] formally investigated inference in MDB. They showed that eliminating precise inference compromise due to functional dependencies and multi-valued dependencies is NP-complete. By extension to this work, the precise elimination of all inferences with respect to the identities of the individuals whose information is included in person-specific data is typically impossible to guarantee. Intuitively this makes sense. Consider two fictitious people named Bob and Alice and Bob is asked to protect his home against invasion from Alice. First, Bob puts locks on his doors and windows. Alice then breaks the glass of a window. Bob responds by installing bars on the windows. Alice now drills through the ceiling. Bob is baffled. The problem is Bob cannot consider a priori every possible attack. This is the case in trying to produce anonymous data as well, so this works seeks to primarily protect against known attacks. As was discussed in chapter 2, the biggest problems result from inferences that can be drawn after linking the released data to other knowledge, so in this work, it is the ability to link the result to foreseeable data sources that must be controlled.

Morgenstern [38] introduced a framework for MDB concerning imprecise inference analysis. His approach involved "spheres of influence" to characterize inference. In comparison to this work, the forward-chained inference process employed in spheres of influence is analogous to linking in this work. That is, Figure 15 could be extended to link more and more data collections beyond the medical data and voter list shown until a chain of links emerged; in this sense, the links extend the sphere. However in this work, attributes are assumed to be independent and only their association with other attributes in a data collection relates them. Morgenstern provides an example in which protecting a person's address should include the person's telephone number because the address can determine the single area code and a limited set of exchanges. Clearly, knowledge from such inferences exploits the semantic relationships between attributes. To combat this problem in this work, I do not require such knowledge be explicitly recognized, but instead rely on the ability to link related attributes. This work assumes related attributes appear in the same collections and in data sources that contain related attributes. For example, phone directories typically contain name, address and phone number as attributes. Therefore, any linking to a phone directory will automatically relate these attributes and protecting one reveals a need to consider the others sensitive.

Catalytic inference analysis was introduced by Hinke [39] and formalized by Hale and Shenoi [40]. Common sense knowledge and discoveries of indirect but related information can provide additional inference when to brought to bear on sensitive information. The approaches taken by Hinke and by Hale and Shenoi are computationally intensive, combating NP-complete problems with dynamic programming used on small data sets. In contrast, this work concerns large and very large databases with algorithms that typically work in real-time. Complexity is substantially reduced by leveraging the fact that the choice of attributes in a collection is an artifact of society and their natural grouping implies a relationship between them [41]. This of course does not capture all the possible ways and kinds of other information that could be brought to bear on the data, which work on catalytic inference analysis attempts to address. In this work, attention is narrowly focused on directly linking data sources using their stated attributes.

Buczkowski [42] used Bayesian probability to estimate security risks due to imprecise inference. In this work however, it is the actual inferred information that is needed and not an estimate of the probability to which a value is inferred.

Many aggregation inference problems can be solved by database design [43, 44], but this solution is not practical in the entity-specific data setting described in chapter 2. In today's environment, information is often divided and partially replicated among multiple data holders and the data holders usually operate autonomously in making disclosure control decisions. The result is that disclosure control decisions are typically made locally with incomplete knowledge of how sensitive other holders of the information might consider replicated data. For example, when somewhat aged information on joint projects is declassified differently by the Department of Defense than by the Department of Energy, the overall declassification effort suffers; using the two partial releases, the original may be reconstructed in its entirety. In general, systems that attempt to produce anonymous data must operate without the degree of omniscience and level of control typically available in the traditional aggregation problem.

In both aggregation and MDB, the primary technique used to control the flow of sensitive information is *suppression*, where sensitive information and all information that allows the inference of sensitive information are simply not released [45]. Suppression can drastically reduce the quality of the data, and in the case of statistical use, overall statistics can be altered, rendering the data practically useless. When protecting national interests, not releasing the information at all may be possible, but the greatest demand for entity-specific data is in situations where the data holder must provide adequate protections while keeping the data useful, such as sharing person-specific medical data for research purposes. In chapters 4 and 5, I will present other techniques and combinations of techniques that produce more useful data than using suppression alone.

## 3.3 Computer security is not privacy protection

An area that might appear to have a common ancestry with disclosure control is access control and authentication, which are traditional areas associated with computer security. Work in this area ensures that the recipient of information has the authority to receive that information. While access control and authentication protections can safeguard against direct disclosures, they do not address disclosures based on inferences that can be drawn from released data. The more insidious problem in disclosure control is not so much whether the recipient can get access or not to the information as much as what values will constitute the information the recipient will receive. A general doctrine of the work presented herein is to release all the information but to do so in a way in which designated properties are protected. Therefore, disclosure control lies outside of traditional work on access control and authentication.

## 3.4 Multiple queries can leak inference

Denning [46] and others [47, 48] were among the first to explore inferences realized from multiple queries to a database. For example, consider a table containing only (*physician*, *patient*, *medication*). A query listing the patients seen by each physician, i.e., a relation R(*physician*, *patient*), may not be sensitive. Likewise, a query itemizing medications prescribed by each physician may also not be sensitive. But the query associating patients with their prescribed medications may be sensitive because medications typically correlate with diseases. One common solution, called *query restriction*, prohibits queries that can reveal sensitive information. This is effectively realized by suppressing all inferences to sensitive data. In contrast, this work poses a real-time solution to this problem by advocating that the data be first rendered sufficiently anonymous, and then the resulting data used as the basis on which queries are processed.

## 3.5 Research on population uniqueness

Skinner and Holmes [49] developed and tested methods for estimating the percent of unique values in the general population based on a smaller database. These methods are based on subsampling techniques and equivalence class structure. Unfortunately, even if these methods provide near-perfect answers they are of limited use in this setting. For example, Figure 16 reports that 12% of the Cambridge voters had unique birth dates. Knowing such underscores the sensitivity of the attribute, but when releasing person-specific information about Cambridge voters, knowing that fact does not help identify which persons in a data collection need their birth date information protected.

## 3.6 Inference, learning and artificial intelligence

Privacy protection, profiling and link analysis have not been traditional areas within artificial intelligence (AI). However, the American Association for Artificial Intelligence held a symposium a couple of years ago to introduce AI researchers to link analysis recognizing that such work could draw on techniques from semantic networks, ontological engineering, graph theory, social network analysis and knowledge discovery in data [50]. These areas, as well as most areas within AI, are concerned with some kind of inference [51]. The best understood is deduction, which logically draws true conclusions from true premises. A second kind of inference is abduction, which is the process of generating explanations from observations and causal relationships. A third kind of inference is induction, which is more commonly known as learning because it occurs when particular examples are used to reach general conclusions. Both abduction and induction can allow false conclusions; nevertheless, they are very useful. While linking data is primarily a deductive process, disclosure control uses all three kinds of inference. Understanding the sensitivity of attributes and the interpretation of associated values often result from abductive and inductive processes.

## 3.7 The *k*-nearest neighbor algorithm

One of the oldest and most analyzed inductive learning procedures is the well-known *k*-nearest neighbor algorithm. Cover and Hart [52] present early theoretical results. Duda and Hart [53] provide a good overview. In this kind of learning method, examples are simply stored as points in *n*-dimensional space. Neighboring points are measured in terms of Euclidian distances between points. The overall space is divided into *k* partitions such that each partition is considered a class. Then, when a new instance is encountered, its relationship to previously stored examples is examined and a classification made based on the Euclidian distance from the new point to neighboring points and therefore, by the division or class in which the new point resides.

While these methods are a cornerstone of the machine learning or knowledge discovery in data field, they have not been used in disclosure control. Yet, such methods could be applied to tabular data. Let each attribute in a table corresponds to a dimension. Let the values themselves, or alternatively the domains of the values, have a numeric presentation with Euclidian properties. Then, a table with *n* attributes and *m* tuples corresponds to *m* points in *n*-dimensional space. The *k*-nearest neighbor algorithm could then be applied, though admittedly, the results would be of limited use, if of any use at all, to disclosure control.

One problem is the number of attributes found in a table. As the number of attributes increases so do the number of dimensions; and, as the number of dimensions increases, finding similarity matches in high dimensional space becomes extremely difficult because of troubles measuring distance. Weights can be applied to each dimension in cases where some dimensions are considered more or less important than others [54]. This equates to lengthening or shortening the axes in Euclidean space. Moore and Lee [55] provide strategies for eliminating the least relevant dimensions from the space. In particular, they provide efficient ways to repeatedly leave one dimension out and then examine the results in order to validate the utility of each dimension.

Another problem concerns the benefit of the results to disclosure control. What is needed is a way to detect the closeness of unusual values in data as the data are being distorted to provide anonymity protection. So in this work, I will present a related algorithm I developed, which I term *k*-Similar, that produces sufficiently anonymous data. This algorithm divides data into groups such that the size of each group consists of *k* or more of the "closest" tuples based on a metric with Euclidian properties.

## Summary

In summary, the catalyst for now examining disclosure control in a broader context has been the dramatic increase in the availability of entity-specific data from autonomous data holders. These changes have expanded the scope and nature of inference control problems and exasperated established operating practice. The goal of this work is to provide comprehensive models for understanding, evaluating and constructing computational systems that control inferences in this setting.

# Chapter 4 Methods

This chapter ends with a formal presentation and real-world systems are evaluated with respect to the formalism in the next chapter. But first, I provide a framework for reasoning about disclosure control and I survey some disclosure limitation techniques using this framework.

## 4.1 Survey of disclosure limitation techniques

I begin by introducing commonly employed disclosure limitation techniques; Figure 17 contains a listing. Here is a quick description of each technique though some were introduced earlier. De-identification [56] and suppression [57] were introduced earlier. *Encryption* is a process of making values secret by replacing one value with another in such a way that certain properties with respect to reversing the process are maintained. *Swapping values* involves exchanging the values associated with an attribute in two tuples where the value from the first tuple becomes the value for the second and vice versa. *Generalization* replaces a value with a more general, less specific alternative. *Substitution* replaces a value with another value in its equivalence class. *Sampling* restricts the number of tuples that will be released. *Scrambling* is a reordering of tuples and is used when the order of appearance of tuples in a release allows inference[1]. Changing *outliers to medians* requires detecting unusual values and replacing them with values that occur more commonly. *Perturbation* involves making changes to values, usually to maintain some overall aggregate statistic. *Rounding* is often used on continuous variables to group values into ranges. Adding *additional tuples* dilutes the number of tuples containing *real* information but values within the newly generated tuples can be chosen to maintain certain aggregate properties. *Additive noise* involves the random incrementing or decrementing of values.

[1] This is slightly inconsistent with the relational model, but in practical use is often an issue.

| | | |
|---|---|---|
| Value and Attribute Based | De-identification | Substitution |
| | Suppression | Outlier to medians |
| | Encryption | Perturbation |
| | Swap values | Rounding |
| | Generalize values | Additive noise |
| Tuple based | Sampling | |
| | Add tuples | |
| | Scramble tuples | |
| Other | Query restriction | |
| | Summaries | |

**Figure 17 Disclosure limitation techniques**

*Query restriction* [58] and *summary data* [59] described earlier are not disclosure limitation techniques but rather special circumstances in which disclosure control is required. In summary data and query restriction, values are often suppressed so as not to reveal sensitive information. This work poses a solution to many problems in query restriction and summarizing by basing queries and summaries on data released from data already determined to be sufficiently anonymous.

*Notice that all of these techniques have the advantage that a recipient of the data can be told what was done to the data in terms of protection*. For data to be useful and results drawn from data to be properly interpreted, it is critical to share what techniques and associated parameters were employed in protecting the confidentiality of entities within the data. Of course usefulness is determined from the point of view of a recipient of the data and what is useful to one recipient is not necessarily beneficial to another. For example, using perturbation can render data virtually useless for learning entity-specific information from the data or identifying entity-specific correlation. On the other hand, using suppression can render data virtually useless for statistical purposes.

During the application of any technique, decisions must be made and these decisions can dramatically impact the data's fitness for a particular purpose. For example, consider a situation in which it is necessary to suppress either values associated with the attribute *ethnicity* or those associated with the attribute *ZIP*. If the recipient of the data is an epidemiologist studying cancer rates near toxic waste sites, then the suppression of *ZIP* may render the data useless. Conversely, if the epidemiologist was studying the prevalence of heart disease among various ethnic groups, then the suppression of *Ethnicity* may have the same ill result. Notice that the data holder cannot release both versions, because doing so may allow the two releases to be linked and reveal all information. Data holders must typically decide a priori for which uses released information will be best suited in order to select the disclosure limitation techniques most appropriate for the task.

## 4.2 Reasoning about disclosure control

The goal of this section is to provide a framework for constructing and evaluating systems that release information such that the released information limits what can be revealed about properties of the entities that are to be protected. For convenience, I focus on person-specific data and the property to be protected is the identity of the subjects whose information is contained in the data. A disclosure implies that an identity was revealed. Consider the informal definition below. Basically, an anonymous data system seeks to effect disclosure control. I use the framework presented in this section to describe the requirements of an anonymous data system and in the next section I formally define such.

### Definition (informal). anonymous data system

> An anonymous data system is one that releases entity-specific data such that particular properties, such as identity, of the entities that are the subject of the data cannot be inferred from the released data.

I can be more specific about how properties are selected and controlled. Recall the real-world examples provided in chapter 2. In those cases, the need for protection centered on limiting the ability to link released information to other external collections. So the properties to be controlled are operationally realized as attributes in the privately held collection. The data holder is expected to identify all attributes in the private information that could be used for linking with external information. Such attributes not only include explicit identifiers such as name, address, and phone number, but also include attributes that in combination can uniquely identify individuals such as birth date and gender. The set of such attributes has been termed a *quasi-identifier* by Dalenius [60] and an *identificate* by Smith [61]. So operationally, an anonymous data system releases entity-specific data such that the ability to link to other information using the quasi-identifier is limited.

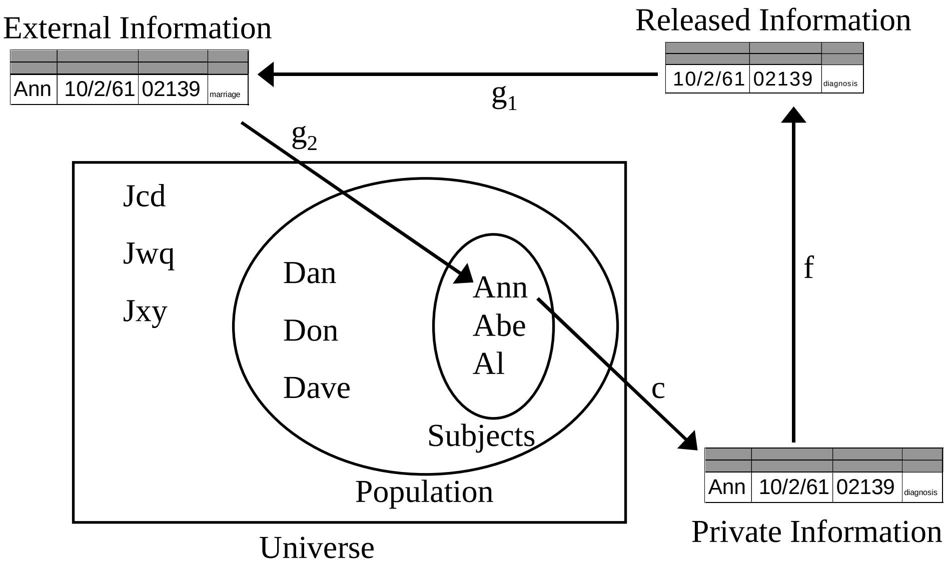


**Figure 18 Release using de-identification**

Figure 18 provides an overview of the disclosure control process. Population consists of persons who are identified as {*Dan*, *Don*, *Dave*, *Ann*, *Abe*, *Al*}, A subset of Population called Subjects is the set of people, in this case {*Ann*, *Abe*, *Al*}, whose information appears in PrivateInformation. Universe consists of Population and the set of *pseudo-entities* {*Jcd*, *Jwq*, *Jxy*}. Pseudo entities are not considered real individuals, as are the members of Population. Instead, the existence of a pseudo-entity is implied by a set of values, which are associated with attributes that identify people, when in fact no such person is associated with that particular set of values.

There exists a collection function *c*: Subjects → PrivateInformation that maps information about members of Subjects into PrivateInformation. The function *f* is a disclosure limitation function such that *f*: PrivateInformation → ReleasedInformation. In the example shown in Figure 18, *f* simply de-identifies tuples from PrivateInformation; and so, the explicit identifier *Ann* is not found in ReleasedInformation.

ExternalInformation results from joining all publicly (and semi-publicly) available information. The relations $g_1$ and $g_2$ illustrate how a tuple in ReleasedInformation can be linked to a tuple in ExternalInformation to re-identify *Ann*, the original subject. *The problem of producing anonymous information can be described as constructing the function f such that some desired invariant exists or some specific assertion can be made about* $g_1$ *and* $g_2$*. Such an invariant or assertion forms the basis for protection.*

In the example shown in Figure 18, the function *f* is simply the de-identification function and the functions $g_1$ and $g_2$ show that *f* is not sufficient; it allows a disclosure. Therefore, merely suppressing explicit identifiers is inadequate.

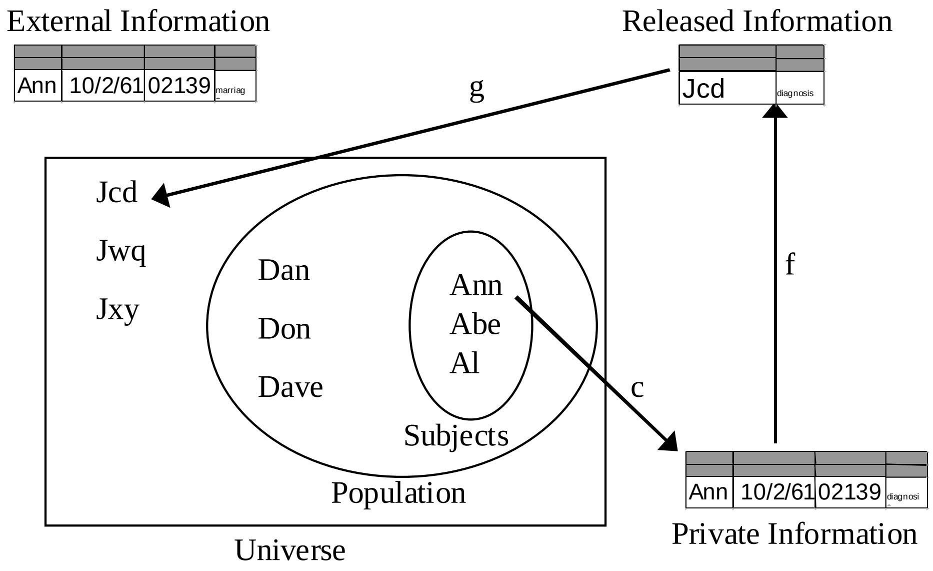


**Figure 19 Release using encryption**

Consider Figure 19. The function *f* seeks to protect the entire quasi-identifier {*name*, *birth date*, *ZIP*} by simply encrypting the associated values. If strong encryption is used and the encrypted values are not used with other releases, then as the diagram in Figure 19 illustrates, the relation *g* will map to a pseudo-entity, being unable to link to ExternalInformation. If on the other hand, *f* used weak encryption then the relation *g* would be able to map directly to *Ann* by simply inverting *f*. Using this approach with strong encryption clearly provides adequate protection, but such protection is at the cost of rendering the resulting information of limited use. Similar results are realized if *f* involved suppression rather than encryption. As shown in Figure 19, the only attribute that remains practically useful is *diagnosis* with no consideration to age or geographical location.

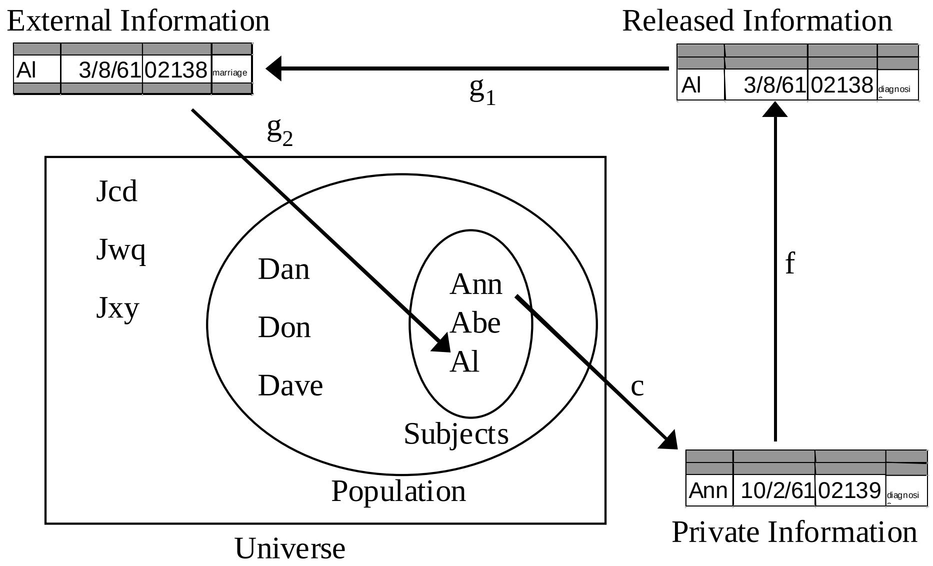


**Figure 20 Release using swapping**

In Figure 20, the function *f* uses swapping [62]. The values associated with the attributes of the quasi-identifier are swapped among tuples. This clearly destroys the integrity of the tuples themselves; however, it maintains overall aggregate statistics. Enforced at the attribute level, this technique can cause extensive distortion. For example if the data are medical information and swapping is employed at the attribute level, a resulting tuple could imply that a 10 year old boy gave birth to a 50 year old woman. Such data would not be very useful for discovering entity-specific patterns pertaining to healthcare cost, outcome or fraud.

A less severe deployment of swapping is shown in Figure 20. In this depiction, the attributes of the quasi-identifier are swapped as a unit among the tuples. A tuple in ReleasedInformation contains the demographic information of *Al* associated with *Ann*'s diagnosis. The relations $g_1$ and $g_2$ show that this tuple can be linked to ExternalInformation because after all, *Al* is a real entity. Suppose *Al*'s original diagnosis involved a cancer whose typical long-term prognosis is excellent, but *Ann*'s diagnosis involved a cancer that is almost always terminal in the short-term. After swapping, *Al* is reported as having the more serious illness. Statisticians who use this technique typically post a notice that warns that the integrity of tuples has been compromised. Even still, the warning usually appears separate and distinct from the data themselves and so, the warning may not be considered during the use of the data and the results can be damaging. For example, the consequences to *Al* in terms of life insurance, employment and credit worthiness may be quite severe and the source of confusion may not be recognized. Also, if the entities whose information is the subject of ReleasedInformation all have cancer, then while a recipient of ReleasedInformation may not know the seriousness of *Al's* cancer, a recipient does know that *Al* has cancer. This underscores an important point. *Implicit attributes often exist in ReleasedInformation and their associated values are the same for all tuples --namely, the identity of the source of the information and the date and time of its creation. Sensitive particulars about the source and/or creation time may be available in ExternalInformation* and therefore allow unwanted inferences.

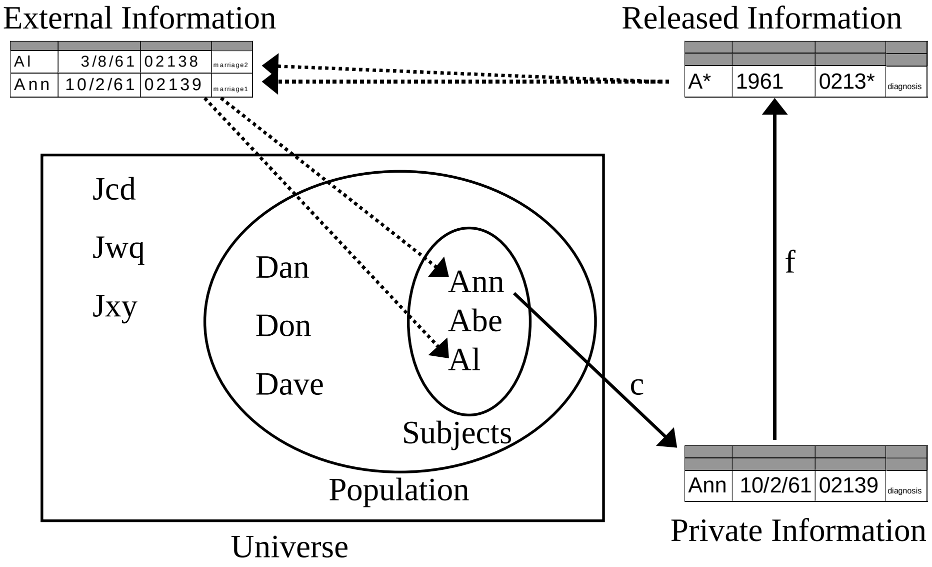


**Figure 21 Release using generalization**

Consider Figure 21. The function *f* generalizes the attributes of the quasi-identifier. I will take a moment to discuss what is meant by generalizing an attribute and then I will return to this scenario for disclosure limitation.

The idea of generalizing an attribute is really a simple concept. A value is simply replaced by a less specific, more general value that is faithful to the original value. In Figure 21 the original ZIP codes {*02138*, *02139*} can be generalized to *0213**, thereby stripping the rightmost digit and semantically indicating a larger geographical area. Likewise {*02141*, *02142*} are generalized to *0214**, and {*0213**, *0214**} could be further generalized to *021***.

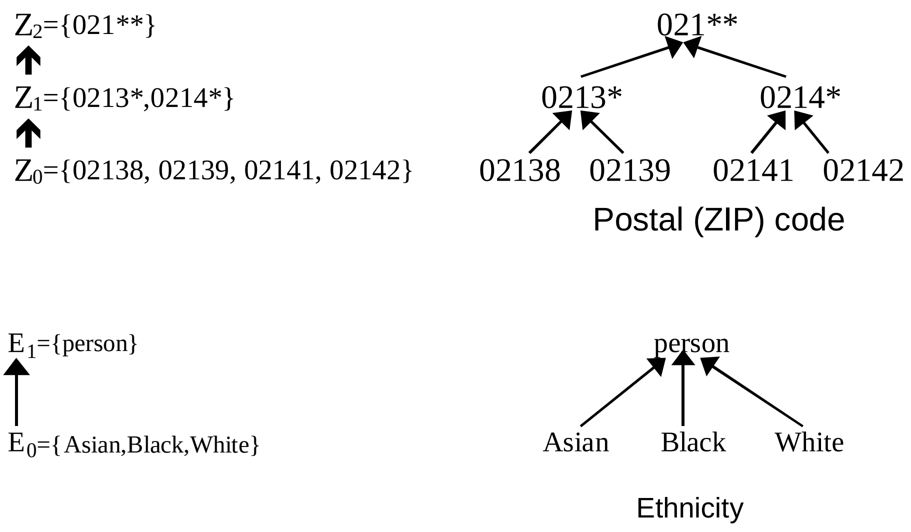


**Figure 22 Generalizing an attribute**

Generalization is effective because substituting values with their more generalized values typically increases the number of tuples having the same values. The single term requirement on the maximal element insures that all values associated with an attribute can eventually be generalized to a single value. All values of all attributes can be semantically organized into generalization hierarchies. Notice in Figure 22 that the values {*Asian*, *Black*, *White*} generalize to *Person*. This means that a generalization of an *Ethnicity* attribute given this hierarchy is similar to suppressing the entire attribute. This demonstrates that generalizing an attribute to its maximal element provides almost the same protection and distortion as suppressing the attribute. The relationship between generalization and suppression will be further discussed in chapter 5.

I now return to Figure 21. The disclosure limitation function *f* generalizes the attributes of the quasi-identifier to produce ReleasedInformation. Tuples in ReleasedInformation can then be linked to ExternalInformation ambiguously. In Figure 21, the tuple shown in ReleasedInformation links to both *Al* and *Ann* in ExternalInformation and so, it relates back to both of them in Subjects. The disclosed *diagnosis* cannot be confidently attributed to either *Al* or *Ann*. In fact, a *k* can be chosen such that *f* generalizes tuples from PrivateInformation in such a way that there are at least *k* possible entities to which each released tuple may refer. Additional protection can often be realized when tuples in ReleasedInformation are ambiguously linked to tuples in ExternalInformation such that the resulting identifications do not only refer to entities in Subjects but also refer to other entities in Universe that are not in Subjects.

A problem however is choosing the right size for *k*. It is based on several parameters including direct and economical communication connections to Subjects. Here is an example. I reviewed some archives from old email exchanges on a newsgroup list and found a couple of email messages pertaining to a chance encounter in Cambridge, Massachusetts between a young woman, whom I will call Alice, and a young man, whom I will call Bob. During the brief conversation between Alice and Bob, no names, addresses or phone numbers were exchanged. Several days later Alice engaged in an email exchange on a newsgroup list in which she provided a casual description of Bob. I constructed a composite of Bob from the email messages. Here is an overview of the details. Bob was about 5’8” in height with dark features. His parents were from Greece. He was believed to live near the water, to enjoy playing soccer and to be an MIT graduate student in electrical engineering or computer science. Given this basic description, I sent a single email message to all members of the electrical engineering and computer science department at MIT. Approximately 1,000 people could have received the message. Five replies were received. All of them had one name, which turned out to be the correct individual. The man himself was quite shocked because he had merely had a private conversation carried in a personal situation and he had not even given his name, phone number, or address. With respect to this disclosure control model, *k* would be about 100 in this case and still that was not sufficient because of the direct and economical communication connection to all-possible subjects and sources of additional information.

This concludes my survey of disclosure limitation techniques and introduction of this framework for reasoning about disclosure control. In the next section I introduce formal models of protection. Following that, I compare and contrast some real-world systems in the next chapter.

## 4.3 Formal protection models

In this section, I formally bring the pieces together; namely, the lessons learned in the real-world examples from chapter 2, the issues presented in the discussion of related work in chapter 3 and the framework for reasoning about disclosure control that was presented earlier in this chapter. Terms mentioned casually and defined informally will be presented formally. So, I begin this section by formally defining the terms I have been using, leading up to the definition of a basic anonymous data system termed **ADS$_0$**. From there, I introduce basic protection models termed *null-map, k-map* and *wrong-map* which provide protection by ensuring that released information maps to no, *k* or incorrect entities, respectively. The non-technical reader may elect to skip this section altogether and continue with the next chapter, which examines four real-world systems that attempt to effect disclosure control.

As stated earlier, I assume the classical relational model of databases [63]. The definition below defines a table and attributes consistent with this model.

### Definition. attributes

Let $\mathsf{B}(A_1,\ldots,A_n)$ be a *table* with a finite number of tuples. The finite set of *attributes* of $\mathsf{B}$ are $\{A_1,\ldots,A_n\}$.

Given a table $\mathsf{B}(A_1,\ldots,A_n)$, $\{A_i,\ldots,A_j\} \subseteq \{A_1,\ldots,A_n\}$, and a tuple $t\in\mathsf{B}$, I use $t[A_i,\ldots,A_j]$ to denote the sequence of the values, $v_i,\ldots,v_j$, of $A_i,\ldots,A_j$ in $t$. I use $\mathsf{B}[A_i,\ldots,A_j]$ to denote the projection, maintaining duplicate tuples, of attributes $A_i,\ldots A_j$ in $\mathsf{B}$.

### Definition. entity

Let $p_i$ = { $(A_i, v_i)$ : $A_i$ is an attribute and $v_i$ is its associated value}. I say $p_i$ is an *entity*. $\mathsf{U}$ = {$p_i$ : $p_i$ is an entity} is a finite set I term a *population of entities.*

### Definition. collection function

Given a population of entities $\mathsf{U}$ and a table $\mathsf{T}$, I say $f_c$ is a collection function on $\mathsf{U}$. That is, $f_c$: $\mathsf{U} \rightarrow \mathsf{T}$ is a *collection function* and $\mathsf{T}$ is an *entity-specific table*. I say that $\mathsf{T}$ is a *person-specific table* if the entities are people.

If $\mathsf{T}$ is an entity specific table containing information about entities in $\mathsf{U}$ and $\mathsf{T}$ contains no additional tuples, then each tuple in $\mathsf{T}$ corresponds to information on at least one entity in $\mathsf{U}$. This is memorialized in the following theorem.

**Theorem 1**

Given a population of entities $\mathsf{U}$, a table $\mathsf{T}(A_1,\ldots,A_n)$, a collection function $f_c$: $\mathsf{U} \rightarrow \mathsf{T}$, and $\{A_i,\ldots,A_j\} \subseteq \{A_1,\ldots,A_n\}$:

$f_c$ is onto $\Rightarrow \forall t[A_i,\ldots,A_j]\in \mathsf{T}$, $\exists p_i\in \mathsf{U}$ such that $\forall(A_x , v_x)\in p_i$ where $A_x\in \{A_i,\ldots,A_j\}$ and $v_x = t[A_x]$.

Proof.

By definition, a function $f_c$ from $\mathsf{U}$ to $\mathsf{T}$ is onto (or a surjection) if and only if for every element in $t\in \mathsf{T}$ there is an element $p\in \mathsf{U}$ with $f_c(p)=t$.

**Example.**

Let $\mathsf{T}$ be a table of visits to a hospital emergency room. Let $\mathsf{U}$ reflect the population of people within the geographical area serviced by the hospital. Then, $f_c$: $\mathsf{U} \rightarrow \mathsf{T}$ is the process for recording hospital visits. Notice that $f_c$ is the collection function and $f_c$ is onto.

## Definition. disclosure control function

Given a table $\mathsf{T}$ and a finite set of tables $\mathsf{B}$, I say $f$ is a disclosure control function on $\{\mathsf{T}\}$. That is, $f$: $\{\mathsf{T}\} \rightarrow \mathsf{B}$ is a *disclosure control function* .

## Definition. re-identification relation

Given a population of entities $\mathsf{U}$, an entity-specific table $\mathsf{T}$ and $f_c$: $\mathsf{U} \rightarrow \mathsf{T}$,

I say $f_g$ is a *re-identification relation* if and only if:

$\exists p_i\in \mathsf{U}$ such that $p_i\in f_g(f_c(p_i))$ and $|f_g(f_c(p_i))| = k$, where $1 \leq k << |\mathsf{U}|$.

I also say that $f_g$ is a re-identification of $p_i$ and I say that $f_g$ uniquely identifies $p_i$ if $k$=1.

*Pseudo entities* are not real entities but their existence is implied by a set of values, one or more of which are false, that are associated with attributes that seem to identify them as entities. This is described in the definition below.

### Definition. pseudo-entities

Given a population of entities $U$, an entity-specific table $T$, $f_c$: $U \rightarrow T$ and a re-identification relation $f_g$: $T \rightarrow U'$ where $U \subseteq U'$. I say $(U'-U)$ is the finite set of *pseudo-entities*.

The following definition formally introduces a quasi-identifier [64] as set of attributes whose associated values may be useful for linking to re-identify the entity that is the subject of the data.

### Definition. quasi-identifier

Given a population of entities $U$, an entity-specific table $T(A_1,\ldots,A_n)$, $f_c$: $U \rightarrow T$ and $f_g$: $T \rightarrow U'$, where $U \subseteq U'$. A quasi-identifier of $T$, written $Q_T$, is a set of attributes $\{A_i,\ldots,A_j\} \subseteq \{A_1,\ldots,A_n\}$ where:

$\exists p_i \in U$ such that $f_g(f_c(p_i)[Q_T]) = p_i$.

**Example.**

Let $V$ be the voter-specific table described earlier in Figure 15 as the voter list. A quasi-identifier for $V$, written $Q_V$, is {*name*, *address*, *ZIP*, *birth date*, *gender*}.

Linking the voter list to the medical data as shown in Figure 15, clearly demonstrates that {*birth date*, *ZIP*, g*ender*} $\subseteq Q_V$. However, {*name*, *address*} $\subseteq Q_V$ because these attributes can also appear in external information and be used for linking.

The goal of disclosure control is to limit the extent to which released information can be confidently linked to other available information. In the case of anonymity, it is usually publicly available data on which linking is to be prohibited and so attributes which appear in private data and also appear in public data are candidates for linking; therefore, these attributes constitute the quasi-identifier and the disclosure of these attributes must be controlled. It is believed that these attributes can be easily identified by the data holder.

**Assumption.**

The data holder can identify attributes in their private information that may also appear in external information.

Consider an instance where this assumption is incorrect; that is, the data holder misjudges which attributes are sensitive for linking. In this case, the released data may be less anonymous than what was required, and as a result, individuals may be more easily identified. Clearly, this risk cannot be perfectly resolved by the data holder because the data holder cannot always know what each recipient of the data knows but policies and contracts can help. Also, the data holder may find it necessary to release data that are only partially anonymous. Again, policies, laws and contracts can provide complementary protections. These are discussed in chapter 6. In the remainder of this work, I assume a proper quasi-identifier has been recognized.

### Definition. explicit-identifier

Let $\mathsf{T}(A_1,\ldots, A_n)$ be a person-specific table and $Q_T(A_i,\ldots,A_j)$ be a quasi-identifier for $\mathsf{T}$. Further, let $\{A_x,\ldots,A_y\} \subseteq Q_T$ and $\mathsf{D}$ be the set of direct communication methods, such as email, telephone, postal mail, etc., where with no additional information, $g_d \in \mathsf{D}$ is a relation from $\mathsf{T}[A_x,\ldots, A_y]$ to the population reachable by $g_d$'s communication method. Let X(s) be a random variable on the sample space s=$\{|g_d( t[A_x,\ldots, A_y])| : t\in \mathsf{T}\}$. I say $\{A_x,\ldots,A_y\}$ is an *explicit identifier* of $\mathsf{T}$ if the expected value of X(s) is 1 and $1/\sigma$ of X(s)$\approx\infty$.

Basically, the definition above states that an explicit identifier is a set of attributes than can be used together with a direct communication method, and no additional information, to distinctly and reliably contact the entity that is the subject of those values for the attributes. Recognizing that such communications are not perfect, the definition implies the method should be almost perfect.

### Definition. explicit-identifiers

Let $\mathsf{T}(A_1,\ldots, A_n)$ be an entity-specific table and $Q_T(A_i,\ldots,A_j)$ be a quasi-identifier for $\mathsf{T}$. The explicit identifiers of $\mathsf{T}$, written, $E_T = \{e_i : e_i$ is an explicit identifier of $\mathsf{T}\}$.

The definition above states that the explicit identifiers of a table is a set of attribute sets, where each member set is an explicit identifier of the table.

**Lemma.**

The explicit identifiers of table T is $E_T$ if and only if the explicit identifiers of a quasi-identifier of $T$ is $E_T$.

**Example**.

The following are examples of explicit identifiers: {*email address*}, {*name*, *address*}, {*name*, *phone number*}. The following are quasi identifiers, but are not explicit identifiers: {*name*}, {*Social Security number*}, {*phone*}, {*phone, Social Security number*}.

Given entity-specific data, an *anonymous data system* releases entity-specific data such that the identities of the entities that are the subject of the original data are protected. Such protection typically relies on a quasi-identifier for the original entity-specific data. The definition below defines a basic anonymous data system.

## Definition. basic anonymous data system

A *basic anonymous data system*, $\mathbf{ADS_0}$, is a nine-tuple (S, P, PT, *QI*, U, R, E, G, *f*), where the following conditions are satisfied:

1. S is the finite set of entities with attributes to be protected.
2. P is the finite set of possible entities. $S \subseteq P$.
3. PT is the finite multi-set of privately held information about each member of S. There exists a collection function, $f_c : S \rightarrow PT$, where $PT=\{k \bullet t_s : t_s = f_c(s) \text{ and } |f_c^{-1}(f_c(s))| = k, \forall s \in S \}$.
4. *QI* is the quasi-identifier of PT denoting attributes to be protected.
5. U is a finite set of possible entities and pseudo-entities. $P \subseteq U$.
6. R is the set of possible releases. Each release RT∈ R is a finite multi-set.
7. E is the collection of possible external information. $\forall T_{i=1,\ldots,m}$ where $T_i$ is a collection of external information about a subset of the members of P, then $E = T_1 \times \ldots \times T_n$.
8. G is the set of possible relations from R → U.

$$G = \left\{ (g_1, g_2) : g_1 \circ g_2 \ where \ R \xrightarrow{g_1} E \xrightarrow{g_2} U \right\}$$

Given a $QI$ for $\mathsf{PT}$, written $QI_{PT}= A_i,\ldots,A_j$, a release $\mathsf{RT}\in \mathsf{R}$ where $\mathsf{RT} = f(\mathsf{PT}[QI])$, and a set of explicit identifiers named $EI_{g2}$ where $g_2(g_1(\mathsf{RT})[EI_{g2}]) \subseteq \mathsf{U}$, then

$g_1(\mathsf{RT}) = \{k \bullet t_u[A_1,\ldots,A_m] : t_u[QI_{PT}] \in \mathsf{RT}, t_u[EI_{g2}] \in \mathsf{E}$ and $|t_u[QI_{PT},,EI_{g2}| = k,$

$\forall t_u\in \mathsf{E}, QI_{PT} \subseteq A_1,\ldots,A_m$ and $EI_{g2} \subseteq A_1,\ldots,A_m \}$.

$g_2$ and $g_2$ are relations and $g_2$ is a direct communication method.

9. $f$ is a disclosure control function such that $f$:{PT}→ R and given a release $\mathsf{RT}\in \mathsf{R}$ where $\mathsf{RT} = f(\mathsf{PT}[QI])$, one of the following conditions must be satisfied:
   a. if $\exists\ g \in \mathsf{G}, \exists\ t \in \mathsf{RT}$, where $f(f_c(s)) = t$ and $g(f(f_c(s))) = s$ then $\exists\ u\in \mathsf{U}$, such that $u \neq s$ and $g(f(f_c(s))) = u$.
   b. if $\exists\ (g_1, g_2) \in \mathsf{G}$ where $\mathsf{GT} = g_1(f(t_s[QI]))$, $\exists\ t_s[QI]\in \mathsf{RT}$ and $t_s[QI, EI_{g2}]\in \mathsf{GT}$ where $f_c(s) = t_s$ and $g_2(g_1(f(t_s[QI]))[EI_{g2}]) = s$, then $\exists\ t_u[QI, EI_{g2}]\in \mathsf{GT}$ such that $t_s\neq t_u$ and $g_2(t_u[QI, EI_{g2}]) = s$.
   c. Given $\mathsf{PT}(A_1,\ldots,A_n)$ and $\mathsf{RT}\ (A_w,\ldots,A_y)$ , let $A_p,\ldots A_q = (\{A_1,\ldots,A_n\} - QI) \cap \{A_w,\ldots,A_y\}$. If $\exists\ g \in \mathsf{G}, \exists\ t_{s1}\ [A_p,\ldots,A_q] \in \mathsf{RT}$, where $f_c(s) = t_{s1}$ and $g(f(t_{s1}[QI])) = s$ and $t_{s1}[A_p,\ldots,A_q]\neq\phi$ and if $\exists\ t_{s2}[A_p,\ldots,A_q]\in \mathsf{PT}$ such that $f_c(s) = t_{s2}$ and $f(t_{s2}) = t_{s1}$ and $t_{s2}[A_p,\ldots,A_q] = t_{s1}[A_p,\ldots,A_q]$, then condition (a) or condition (b) above must be satisfied on $t_{s1}$.

The overall concept is of an anonymous data system is that a derivate of privately collected data are released such that the subjects of the data cannot be confidently or uniquely identified.

The main property is property 9. It says that if $f$ produces a release $\mathsf{RT}\in \mathsf{R}$ based on $\mathsf{PT}[QI]$, then there can not exist a function or composite of functions which can confidently associate any of the original subjects uniquely with their information in PT.

If an entity is correctly associated with a released tuple in RT, then the three conditions required in property 9 are: (1) there must be more than one such entity to which the tuple in the release could be associated; (2) there must be more than one such tuple in the release that could be associated with the subject; or, (3) the non-controlled information, if present, can not be accurate.

Properties 3, 7 and 8 describe multiset collections of information where collections of elements can occur as a member more than once.

The definition above describes what is termed a *basic anonymous data system*. The word "basic" is used and the subscript 0 attached because the definition does not allow for probabilistic linking or the temporal nature of data quality (i.e., older data can be less reliable). For anonymous data systems to be defined to include these issues requires a modification and extension to $\mathbf{ADS_0}$ and so, the naming convention reserves $\mathbf{ADS_1}$ and $\mathbf{ADS_2}$ and so on, for future enhancements.

**Remark.**

The level of protection provided by an $\mathbf{ADS_0}$ depends on the correctness of the selection of attributes within *QI*, on the specifics of *f* and on assertions and invariants that can be made about $g_1$ and $g_2$, $\forall(g_1, g_2)\in$ G. The validity of this remark stems directly from the definition of an $\mathbf{ADS_0}$.

S = {(*name*, Ann), (*name*, Abe), (*name*, Al)}
P = {(*name*, Dan), (*name*, Don), (*name*, Dave), (*name*, Ann), (*name*, Abe), (*name*, Al)}
PT(*name*, *birth date*, *ZIP*, *diagnosis*) **:**

| Name | Birth date | ZIP | Diagnosis |
|---|---|---|---|
| Ann | 10/2/61 | 02139 | Cardiac |
| Abe | 7/14/61 | 02139 | Cancer |
| Al | 3/8/61 | 02138 | Liver |

*QI* = {*name*, *birth date*, *ZIP*}
U = {(*name*, Jcd), (*name*, Jwq), (*name*, Jxy), (*name*, Dan), (*name*, Don), (*name*, Dave),
(*name*, Ann), (*name*, Abe), (*name*, Al) }
*E*(*name*, *birth date*, *ZIP*) **:**

| Name | Birth date | ZIP |
|---|---|---|
| Ann | 10/2/61 | 02139 |
| Abe | 7/14/61 | 02139 |
| Al | 3/8/61 | 02138 |

$g_2$ = a direct communication channel that operates on the *name* attribute.
G as the set of all possible relations from R to U consistent with property 8 in the definition of an $\mathbf{ADS_0}$

**Figure 23 Values for S, P, PT, *QI*, U and E**

In the following examples, I assume the values for S, P, PT, *QI*, U, and E shown in Figure 23. These values are consistent with the presentations in Figure 18, Figure 19, Figure 20 and Figure 21.

**Example (identity release)**.

Given the assignments in Figure 23, and the following definition for *f* that constructs RT as a copy of PT, the system **A**(S, P, PT, *QI*, U, {RT}, E, G, *f*) is not an $\mathbf{ADS_0}$.

*f* is defined as follows:

```
step 1.  Let RT be $\emptyset$
step 2.  $\forall t \in$ PT, RT $\leftarrow$ RT $\cup$ $\{t\}$
```

Note. RT is a multi-set, so duplicates are maintained.

Proof:

Let $g_1$ be the relation $g_1$(*name*, *birth date*, *ZIP*, *diagnosis*) on RT.

Therefore **A** is insecure and a disclosure is made, so **A** is not an $\mathbf{ADS_0}$.

**Example (complete suppression)**.

Given the definitions in Figure 23, and the following definition for *f* that constructs RT as a blank table, the system **A**(S, P, PT, *QI*, U, {RT}, E, G, *f*) is an $\mathbf{ADS_0}$.

*f* is defined as follows:

```
step 1.  Let RT be $\emptyset$
step 2.  $\forall t \in$ PT, RT $\leftarrow$ RT $\cup$ {null, null, null, null}
```

Note. RT is a multi-set, so duplicates are maintained.

Proof:

The first two conditions of property 9 in the definition of an $\mathbf{ADS_0}$ are both satisfied $\forall t \in$ RT.

Therefore **A** is considered secure, so **A** is an $\mathbf{ADS_0}$.

The two examples above demonstrate the natural tension that exists in disclosure control. At one end is specificity and usefulness, which is not secure, and at the other end is distortion and security, which is not useful. These opposites pose a continuum of disclosure control options along which tradeoffs must be made. I defined and used an information theoretic (entropy) metric [65] and measured the distortion to data caused by common disclosure limitation techniques and then plotted the measures along the continuum. The relative ordering of the results is shown below in Figure 24.

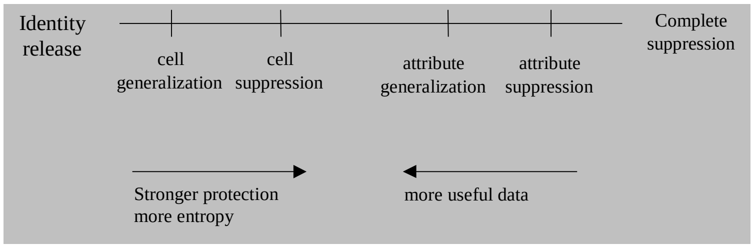


**Figure 24 Relative comparison of techniques**

The technique *cell generalization* is generalization enforced at the cell level and likewise *cell suppression* is suppression enforced at the cell level. Similarly, *attribute generalization* is generalization enforced at the attribute level and *attribute suppression* is suppression enforced at the attribute level. Do not interpret the tick marks along the continuum as points. Each of these techniques had results in a range along the continuum and the ranges overlapped; further there was significant variation depending on the character of the data. However, the tick marks do provide a relative ordering of the medians of average case results.

I now present three protection models for $\mathbf{ADS_0}$. These are *wrong-map*, *null-map* and *k-map* as defined below.

### Definition. null-map protection

Let **A** be an $\mathbf{ADS_0}$, $f(\mathrm{PT}) = \mathrm{RT}$ and $\mathrm{R} \in \mathrm{RT}$. If $\forall t \in \mathrm{RT}$, there does not exist $g \in \mathrm{G}$ where $g(t) \in \mathrm{S}$, then **A** adheres to *null map protection*.

In *null-map protection* each tuple in the released information may or may not map to an actual entity in the population P, but none of the tuples can be mapped to an entity in the set of subjects S. Examples of disclosure limitation techniques that can achieve null-map protection include strong encryption of the *QI*, extensive swapping of the values in *QI* and systematic use of additive noise. Figure 19 provides an example.

### Definition. wrong-map protection

Let **A** be an $\mathbf{ADS_0}$, $f(\mathrm{PT}) = \mathrm{RT}$ and $\mathrm{R} \in \mathrm{RT}$. If $|\mathrm{RT}| > 2$ and $\forall t \in \mathrm{RT}$, $\exists g \in \mathrm{G}$ where $f(f_c(s)) = t$, and $g(f(f_c(s))) = s$ and there does not exist $g' \in \mathrm{G}$ where $g' \neq g$ such that $g'(t) \in \mathrm{S}$, then **A** adheres to *wrong map protection*.

*Wrong map protection* requires each tuple in the released information to be identified to only one entity in subjects but that entity is not the entity to which the original information was collected. The $\mathbf{ADS_0}$ requirement ensures the values with attributes outside *QI* contained in the release are not the same as those originally collected. Notice if there exists only one entity in the subjects S, then wrong-map protection cannot be done and with only two entities in S, the release is compromised. An example of a disclosure limitation technique that can achieve wrong map protection is swapping the attributes of *QI* as a unit. Figure 20 provides an example.

### Definition. *k*-map protection

Let **A** be an $\mathbf{ADS_0}$, $f(\mathrm{PT}) = \mathrm{RT}$ and $\mathrm{R} \in \mathrm{RT}$. If$\forall t \in \mathrm{RT}$, $\exists\ g \in \mathrm{G}$, where $f(f_c(s)) = t$ and $g(f(f_c(s))) = s$ and $\{u_1, u_2, u_{k-1}\} \in \mathrm{U}$ such that for $i=1,\ldots, k-1$, $u_i \neq s$, and $g(f(f_c(s))) = u_i$, then **A** adheres to *k-map protection*.

*k-map protection* maintains the invariant that each tuple in the released information refers indistinctly to at least *k* members of U. Notice that *k* does not rely on $|\mathrm{S}| > k$ or on $|\mathrm{RT}| > k$. Figure 21 provides an example.

The protection models *k*-map, null-map and wrong-map provide a means for characterizing the kind of protection provided to a release of information. Of course a release may be anonymous, but proving it in the absence of a protection model is extremely difficult. Optimal releases that offer adequate protection with minimal distortion are believed to typically require a combination of disclosure limitation techniques as well as a combination of protection models.

## 4.4 Future work

1. The protection models defined in this chapter, namely, *k*-map, wrong-map and null-map, are not necessarily a complete set of all possible protection models. Develop a new protection model or compare and contrast the relative protection provided by each of these models.

2. Recall the word "basic" is used and the subscript 0 attached to a basic anonymous data system ($\mathbf{ADS_0}$) because the definition does not allow for probabilistic linking or the temporal nature of data quality (i.e., older data can be less reliable). For anonymous data systems to be defined to include these issues requires a modification and extension to $\mathbf{ADS_0}$ and so, the naming convention reserves $\mathbf{ADS_1}$ and $\mathbf{ADS_2}$ and so on, for future enhancements. Extend $\mathbf{ADS_0}$ along these lines.

# Chapter 5 Methods Extended – Preferred Minimal Generalization Algorithm

The goal of this chapter is to extend the formal methods provided in the previous chapter and formally present an algorithm that adheres to *k*-map protection using generalization and suppression. The real-world systems Datafly [66], μ-Argus [67] and *k*-Similar [68] motivate this extension.

## 5.1 The *k*-anonymity protection model

As you may recall, the *k*-map protection model [69] states an anonymity constraint that requires certain characteristics and combinations of characteristics found in the data to combine to match at least *k* individuals. To determine how many individuals each released tuple actually matches requires combining the released data with externally available data and analyzing other possible attacks. Making such a determination directly can be an impossible task for the data holder who releases information. Although I can assume the data holder knows which data in PT also appear externally, and therefore what constitutes a quasi-identifier, the specific values of external data and knowledge of other possible inference attacks cannot be assumed. I therefore seek to protect the information by satisfying a slightly different constraint on released data, which I term the *k-anonymity* requirement. This is a special case of *k*-map protection where *k* is enforced on the released data.

**Definition. *k*-anonymity**

Let RT($A_1$,...,$A_n$) be a table and $QI_{RT}$ be the quasi-identifier associated with it. RT is said to satisfy *k*-anonymity if and only if each sequence of values in RT[$QI_{RT}$] appears with at least *k* occurrences in RT[$QI_{RT}$].

| | Ethnicity | Birth | Gender | ZIP | Problem |
|---|---|---|---|---|---|
| t1 | Black | 1965 | m | 0214* | short breath |
| t2 | Black | 1965 | m | 0214* | chest pain |
| t3 | Black | 1965 | f | 0213* | hypertension |
| t4 | Black | 1965 | f | 0213* | hypertension |
| t5 | Black | 1964 | f | 0213* | obesity |
| t6 | Black | 1964 | f | 0213* | chest pain |
| t7 | White | 1964 | m | 0213* | chest pain |
| t8 | White | 1964 | m | 0213* | obesity |
| t9 | White | 1964 | m | 0213* | short breath |
| t10 | White | 1967 | m | 0213* | chest pain |
| t11 | White | 1967 | m | 0213* | chest pain |

**Figure 25 Example of *k*-anonymity, where *k*=2 and QI={*Ethnicity*, *Birth*, *Gender*, *ZIP*}**

**Example.**

Figure 25 provides an example of a table T that adheres to *k*-anonymity. The quasi-identifier for the table is $QI_T$= {*Ethnicity*, *Birth*, *Gender*, *ZIP*} and *k*=2. Therefore, for each of the tuples contained in the table T, the values of the tuple that comprise the quasi-identifier appear at least twice in T. That is, for each sequence of values in T[$QI_T$] there are at least *2* occurrences of those values in T[$QI_T$]. In particular, *t1*[$QI_T$] = *t2*[$QI_T$], *t3*[$QI_T$] = *t4*[$QI_T$], *t5*[$QI_T$] = *t6*[$QI_T$], *t7*[$QI_T$] = *t8*[$QI_T$] = *t9*[$QI_T$], and *t10*[$QI_T$] = *t11*[$QI_T$].

**Lemma.**

Let RT($A_1$,...,$A_n$) be a table, $QI_{RT}$ =($A_i$,…, $A_j$) be the quasi-identifier associated with RT, $A_i,\ldots,A_j \subseteq A_1,\ldots,A_n$, and RT satisfy *k*-anonymity. Then, each sequence of values in RT[$A_x$] appears with at least *k* occurrences in RT[$QI_{RT}$] for $x=i,\ldots,j$.

**Example.**

Figure 25 provides an example of a table T that adheres to *k*-anonymity. The quasi-identifier for the table is $QI_T$= {*Ethnicity*, *Birth*, *Gender*, *ZIP*} and *k*=2. Therefore, each value that appears in a value associated with an attribute of QI in T appears at least *k* times. |T[*Ethnicity* ="black"]| = 6. |T[*Ethnicity* ="white"]| = 5. |T[*Birth* ="1964"]| = 5. |T[*Birth* ="1965"]| = 4. |T[*Birth* ="1967"]| = 2. |T[*Gender* ="m"]| = 6. |T[*Gender* ="f"]| = 5. |T[*ZIP* ="0213*"]| = 9. And, |T[*ZIP* ="0214*"]| = 2.

It can be trivially proven that if the released data RT satisfies *k*-anonymity with respect to the quasi-identifier $QI_{PT}$, then the combination of the released data RT and the external sources on which $QI_{PT}$ was based, cannot link on $QI_{PT}$ or a subset of its attributes to match fewer than *k* individuals. This property holds provided that all attributes in the released table RT which are externally available in combination (i.e., appearing together in an external table or in a possible join of external tables) are defined in the quasi-identifier $QI_{PT}$ associated with the private table PT. This property does not guarantee individuals cannot be identified in RT; there may exist other inference attacks that could reveal the identities of the individuals contained in the data. However, the property does protect RT against inference from linking to known external sources; and in this context, the solution can provide an effective guard against re-identifying individuals.

As an aside, there are many ways in which I could expand the notion of a quasi-identifier to provide more flexibility and granularity. Both Datafly [70] and μ-Argus [71] weight the attributes of the quasi-identifier. For my purposes in this chapter, however, I begin by considering a single quasi-identifier based on attributes, without weights, appearing together in an external table or in a possible join of external tables; and then later in this chapter, I add weights to specify preferences among the attributes of the quasi-identifier.

## 5.2 Generalization and suppression as disclosure limitation techniques

In this section, I formally present the disclosure limitation techniques known as generalization [72] and suppression [73]. This chapter ends by my proposing an algorithm that produces a version of PT such that a given *k*-anonymity requirement is satisfied by re-coding values to make them more general (i.e., using generalization and suppression).

In a classical relational database system, domains are used to describe the set of values that attributes assume. For example, there might be a ZIP code domain, a *number* domain and a *string* domain. I extend this notion of a domain to make it easier to describe how to generalize the values of an attribute. In the original database, where every value is as specific as possible, every attribute is considered to be in a ground domain. For example, 02139 is in the ground ZIP code domain, $Z_0$. In order to achieve *k*-anonymity I can make ZIP codes less informative. I do this by saying that there is a more general, less specific domain that can be used to describe ZIP codes, say $Z_1$, in which the last digit has been replaced by 0 (or removed altogether). There is also a mapping from $Z_0$ to $Z_1$, such as 02139 → 02130.

Given an attribute *A*, I say a *generalization for an attribute* is a function on *A*. That is, each *f*: *A* → *B* is a generalization. I also say that:

$$A_o \xrightarrow{f_0} A_1 \xrightarrow{f_1} \dots \xrightarrow{f_{n-1}} A_n$$

is a generalization sequence or a functional generalization sequence.

Given an attribute *A* of a private table PT, I define a **domain generalization hierarchy** $\mathrm{DGH}_A$ for *A* as a set of functions $f_h$ : *h*=0,…,*n*-1 such that:

$$A_o \xrightarrow{f_0} A_1 \xrightarrow{f_1} \dots \xrightarrow{f_{n-1}} A_n$$

$A=A_0$ and $|A_n| = 1$. $\mathrm{DGH}_A$ is over: $\bigcup_{h=0}^{n} A_h$

Clearly, the $f_h$'s impose a linear ordering on the $A_h$'s where the minimal element is the ground domain $A_0$ and the maximal element is $A_n$. The singleton requirement on $A_n$ ensures that all values associated with an attribute can eventually be generalized to a single value. Since generalized values are used in place of more specific ones, it is important that all domains in the hierarchy be compatible. Using the same storage representation form for all domains in the generalization hierarchy can ensure compatibility. In my ZIP code example above, replacing the last digit with 0, rather than removing it or changing it to *, maintains the 5 digit storage representation. In this presentation I assume $A_h$, *h*=0,…,*n*, are disjoint; if an implementation is to the contrary and there are elements in common, then $\mathrm{DGH}_A$ is over the disjoint sum of $A_h$'s and subsequent definitions change accordingly.

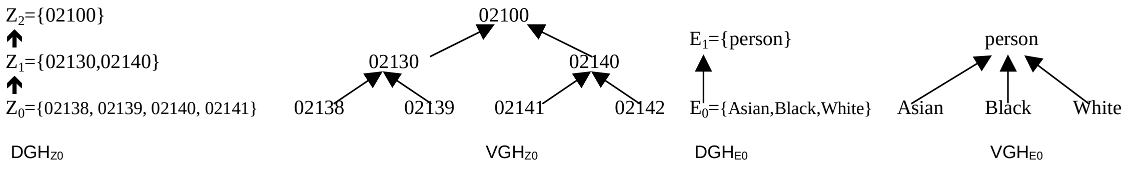


Figure 26 Examples of domain and value generalization hierarchies

| Eth:$E_0$ | ZIP:$Z_0$ |
|---|---|
| Asian | 02138 |
| Asian | 02139 |
| Asian | 02141 |
| Asian | 02142 |
| Black | 02138 |
| Black | 02139 |
| Black | 02141 |
| Black | 02142 |
| White | 02138 |
| White | 02139 |
| White | 02141 |
| White | 02142 |

PT

| Eth:$E_1$ | Zip:$Z_0$ |
|---|---|
| Person | 02138 |
| Person | 02139 |
| Person | 02141 |
| Person | 02142 |
| Person | 02138 |
| Person | 02139 |
| Person | 02141 |
| Person | 02142 |
| Person | 02138 |
| Person | 02139 |
| Person | 02141 |
| Person | 02142 |

$GT_{[1,0]}$

| Eth:$E_1$ | ZIP:$Z_1$ |
|---|---|
| Person | 02130 |
| Person | 02130 |
| Person | 02140 |
| Person | 02140 |
| Person | 02130 |
| Person | 02130 |
| Person | 02140 |
| Person | 02140 |
| Person | 02130 |
| Person | 02130 |
| Person | 02140 |
| Person | 02140 |

$GT_{[1,1]}$

| Eth:$E_0$ | ZIP:$Z_2$ |
|---|---|
| Asian | 02100 |
| Asian | 02100 |
| Asian | 02100 |
| Asian | 02100 |
| Black | 02100 |
| Black | 02100 |
| Black | 02100 |
| Black | 02100 |
| White | 02100 |
| White | 02100 |
| White | 02100 |
| White | 02100 |

$GT_{[0,2]}$

| Eth:$E_0$ | ZIP:$Z_1$ |
|---|---|
| Asian | 02130 |
| Asian | 02130 |
| Asian | 02140 |
| Asian | 02140 |
| Black | 02130 |
| Black | 02130 |
| Black | 02140 |
| Black | 02140 |
| White | 02130 |
| White | 02130 |
| White | 02140 |
| White | 02140 |

$GT_{[0,1]}$

Figure 27 Examples of generalized tables for PT

Given a domain generalization hierarchy $DGH_A$ for an attribute *A*, if $v_i \in A_i$ and $v_j \in A_j$ then I say $v_i \leq v_j$ if and only if $i \leq j$ and:

$$f_{j-1}(\ldots f_i(v_i)\ldots) = v_j$$

This defines a *partial ordering* ≤ on: $\bigcup_{h=0}^{n} A_h$

Such a relationship implies the existence of a **value generalization hierarchy** $VGH_A$ for attribute *A*. Figure 26 illustrates an example of domain and value generalization hierarchies for domain $Z_0$, representing ZIP codes for Cambridge, MA, and $E_0$ representing ethnicity.

### 5.2.1 Generalization including suppression

In the value generalization hierarchy $VGH_{E0}$ shown in **Figure 26**, the values {*Asian*, *Black*, *White*} generalize to *Person*. This means that a generalization of *Ethnicity* is similar to suppressing that value for the attribute. Generalizing an attribute to its maximal element provides almost the same protection and distortion as suppressing the attribute.

Therefore, I can expand my presentations of generalization to include suppression by imposing on each value generalization hierarchy a new maximal element, atop the old maximal element. The new maximal element is the attribute's suppressed value. The height of each value generalization hierarchy is therefore incremented by one. No other changes are necessary to incorporate suppression into the earlier presentation of generalization. Figure 28 and Figure 29 provide examples of the domain and value generalization hierarchies shown earlier in **Figure 26**, but expanded here to include the suppressed maximal element. From now on, all references to generalization include the new maximal element atop each domain and value generalization hierarchy.

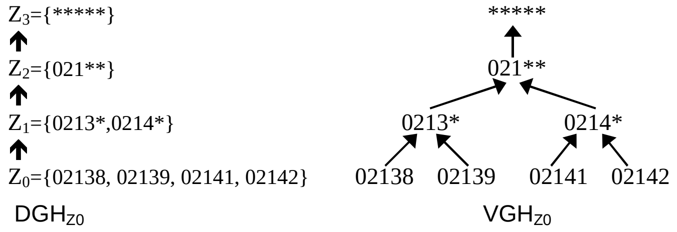


**Figure 28 ZIP domain and value generalization hierarchies including suppression**

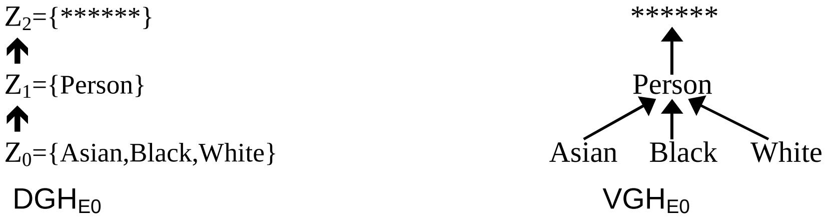


**Figure 29 Ethnicity domain and value generalization hierarchies including suppression**

## 5.3 Minimal generalization of a table

Given a private table PT, generalization can be effective in producing a table RT that is based on PT but that adheres to *k*-map protection because values in RT are substituted with their generalized replacements. The number of distinct values associated with each attribute is non-increasing, and so the substitution tends to map values to the same generalized result, thereby possibly decreasing the number of distinct tuples in RT.

A generalization function on tuple $t$ with respect to $A_1,\ldots, A_n$ is a function $f_t$ on $A_1\times\ldots\times A_n$ such that:

$$f_t(A_1,\ldots,A_n)=(f_{t1}(A_1),\ldots,f_{tn}(A_n))$$

where for each $i$: $1,\ldots,n$, $f_{ti}$ is a generalization of the value $t[A_i]$. The function $f_t$ is a set function. I say $f_t$ is generated by the $f_{ti}$'s.

Given $f$, $A_1,\ldots,A_n$, a table $T(A_1,\ldots,A_n)$ and a tuple $t\in T$, i.e., $t(a_1,\ldots,a_n)$

$$g(T) = \left\{ k \cdot f(t) : t \in T \; and \left| f^{-1}(f(t)) \right| = k \right\}$$

The function *g* is a multi-set function. I say that *g* is the multi-set function generated by *f* and by the $f_i$'s. Further, I say that *g(*T*) is a generalization of table* T. This does not mean, however, that the generalization respects the value generalization hierarchy for each attribute in T. To determine whether one table is a generalization with respect to the value generalization hierarchy of each attribute requires analyzing the values themselves.

Let $\mathsf{DGH}_i$ be the domain generalization hierarchies for attributes $A_{li}$ where $i$=1,…,$A_n$ Let $\mathsf{T}_l[A_{l1},\ldots,A_{lAn}]$ and $\mathsf{T}_m[A_{m1},\ldots,A_{mAn}]$ be two tables such that for each $i$:1,..,$n$, $A_{li},A_{mi} \in \mathsf{DGH}_i$. Then, I say table $\mathsf{T}_m$ is **a generalization of table** $\mathsf{T}_l$, written $\mathsf{T}_l \le \mathsf{T}_m$, if and only if there exists a generalization function *g* such that $g[\mathsf{T}_l] = \mathsf{T}_m$ and is generated by $f_i$'s where: $\forall t_l \in \mathsf{T}_l$, $a_{li} \le f_i(a_{li}) = a_{mi}$ *and* $f_i : A_{li} \rightarrow A_{mi}$ and each $f_i$ is in the $\mathsf{DGH}_i$ of attribute $A_{li}$. From this point forward, I will use the term *generalization* to denote a generalization of a table. Otherwise, I will explicitly refer to the set or multi-set function when it is not otherwise clear from context.

In this work, I examine cell, or value-level generalization, as well as, generalization enforced at the attribute level. When decisions about the values an attribute can assume are specific to a single domain – that is, each value associated with an attribute in a table must be a member of the same domain in the domain generalization hierarchy specific to that attribute -- then I say the decision is at the attribute level. On the other hand, if different values associated with the same attribute in a table can have different domains in the domain generalization hierarchy specific to that attribute, then I say the decision is at the cell or value level.

## Definition. *k*-anonymity requirement

Let $\mathsf{T}(A_1,\ldots,A_n)$ be a generalized table, $\mathsf{QI}_\mathsf{T}=\{A_i,\ldots,A_j\}$ be the quasi-identifier associated with it where $\{\mathrm{A}_\mathrm{i},\ldots,\mathrm{A}_\mathrm{j}\} \subseteq \{A_1,\ldots,A_n\}$, $\mathrm{t} \in \mathsf{T}[\mathsf{QI}_\mathsf{T}]$ and $k_t$ be the integer denoted in *g* for *f*(*t*). T is said to satisfy a *k*-anonymity requirement of *k* with respect to $QI_T$ if $\forall t \in \mathsf{T}[\mathsf{QI}_\mathsf{T}]$, $k_t \ge k$.

The *k*-anonymity requirement of a generalized table forms the basis for *k*-map protection. Given a table $\mathsf{PT}(A_1,\ldots,A_n)$, a $\mathsf{QI}_{\mathsf{PT}}=\{A_i,\ldots,A_j\}$, where $\{A_i,\ldots,A_j\} \subseteq \{A_1,\ldots,A_n\}$, and a generalization of PT with respect to $\mathsf{QI}_{\mathsf{PT}}$ named $\mathsf{RT}(A_i,\ldots,A_j)$, satisfaction of the *k*-anonymity requirement guarantees each $t \in \mathsf{RT}$ is indistinguishable from at least *k-1* other members of table $\mathsf{RT}[\mathsf{QI}_{\mathsf{PT}}]$.

**Example**

Consider the table PT illustrated in **Figure 27** and the domain and value generalization hierarchies for $\mathsf{E}_0$ and $\mathsf{Z}_0$ illustrated in Figure 28 and Figure 29. The remaining four tables in the figure are examples of generalized tables for PT where generalization is enforced at the attribute level. For the clarity of the example, every table reports, together with each attribute, the domain for the attribute in the table. With respect to *k*-anonymity: $\mathsf{GT}_{[0,1]}$ satisfies *k*-anonymity for $k$ = 1,2; $\mathsf{GT}_{[1,0]}$ satisfies *k*-anonymity for $k$ = 1, 2, 3; $\mathsf{GT}_{[0,2]}$ satisfies *k*-anonymity for $k$ = 1,...,4; and, $\mathsf{GT}_{[1,1]}$ satisfies *k*-anonymity for $k$ = 1,...,6.

It is easy to see that the number of different domain generalizations of a table T, when generalization is enforced at the attribute level, is equal to the number of different combinations of domains that the attributes in the table can assume. Suppose I have domain generalization hierarchies $\mathsf{DGH}_{\mathsf{i}}$ for $A_i$, $i$:1,…,$n$; then, the number of generalizations, enforced at the attribute level, for table $\mathsf{T}(A_1,\ldots,A_n)$ is:

$$\prod_{i=1}^{n}\left(\left|\mathsf{DGH}_i\right|+1\right)$$

**Equation 1**

Similarly, when generalization is enforced at the cell level, the number of different generalizations of a table T is equal to the number of different combinations of values the cells within T can assume. Given domain generalization hierarchies $\mathsf{DGH}_{\mathsf{i}}$ for $A_i$, $i$:1,…,$n$; then, the number of generalizations, enforced at the cell level, for table $\mathsf{T}(A_1,\ldots,A_n)$ is:

$$\prod_{i=1}^{n}\left(\left|\mathsf{DGH}_i\right|+1\right)^{|\mathsf{T}|}$$

**Equation 2**

Clearly, not all such generalizations are equally satisfactory. A trivial possible generalization, for instance, is the one that generalizes each attribute to the highest possible level of generalization, thus collapsing all tuples in the table to the same list of values. This provides *k*-anonymity at the price of a strong generalization of the data. Such extreme generalization is not needed if a less generalized table (i.e., containing more specific values) exists which satisfies *k*-anonymity. This concept is captured by the following definition of *k*-minimal generalization.

### Definition. *k*-minimal generalization

Let $\mathsf{T}_l(A_1,\ldots,A_n)$ and $\mathsf{T}_m(A_1,\ldots,A_n)$ be two tables such that $\mathsf{T}_l[\mathsf{QI}_\mathsf{T}] \leq \mathsf{T}_m[\mathsf{QI}_\mathsf{T}]$, where $\mathsf{QI}_\mathsf{T}=\{A_i,\ldots,A_j\}$ is the quasi-identifier associated with the tables and $\{A_i,\ldots,A_j\} \subseteq \{A_1,\ldots,A_n\}$. $\mathsf{T}_m$ is said to be a minimal generalization of a table $\mathsf{T}_l$ with respect to a *k* anonymity requirement over $\mathsf{QI}_\mathsf{T}$ if and only if:

1. $\mathsf{T}_m$ satisfies the *k*-anonymity requirement with respect to $\mathsf{QI}_\mathsf{T}$
2. $\forall \mathsf{T}_z$: $\mathsf{T}_l \leq \mathsf{T}_z$, $\mathsf{T}_z \leq \mathsf{T}_m$, $\mathsf{T}_z$ satisfies the *k*-anonymity requirement with respect to $\mathsf{QI}_\mathsf{T} \Rightarrow$ $\mathsf{T}_z[A_1,\ldots,A_n] = \mathsf{T}_m[A_1,\ldots,A_n]$.

**Example.**

**Figure 27** shows examples of generalizations of the table labeled PT with respect to the quasi-identifier {*Ethnicity*, *ZIP*}. Each of these generalizations, enforced at the attribute level, satisfy *k*-anonymity for *k*=2. That is, each tuple in the released tables, labeled $\mathsf{GT}_{[1,0]}$, $\mathsf{GT}_{[1,1]}$, $\mathsf{GT}_{[0,2]}$, $\mathsf{GT}_{[0,1]}$, appears at least 2 times. $\mathsf{GT}_{[0,1]}$ shows that generalizing *ZIP* one level up its domain generalization hierarchy is sufficient to achieve *k*=2. Similarly, $\mathsf{GT}_{[1,0]}$ shows that generalizing *Ethnicity* one level up its domain generalization hierarchy is sufficient to achieve *k*=2. Therefore, $\mathsf{GT}_{[1,1]}$, and $\mathsf{GT}_{[0,2]}$ perform more generalization than is necessary, because table $\mathsf{GT}_{[0,2]}$, which satisfies the anonymity requirement, is a generalization of $\mathsf{GT}_{[0,1]}$. Analogously, $\mathsf{GT}_{[1,1]}$ cannot be minimal, being a generalization of both $\mathsf{GT}_{[1,0]}$ and $\mathsf{GT}_{[0,1]}$. Further, because both $\mathsf{GT}_{[0,1]}$ and $\mathsf{GT}_{[1,0]}$ satisfy the requirement and are minimal, there may exist a preference among these minimal generalizations.

Intuitively, a table $\mathsf{T}_m$, generalization of $\mathsf{T}_l$, is *k*-minimal if it satisfies *k*-anonymity and there does not exist any generalization of $\mathsf{T}_l$, which satisfies *k*-anonymity and of which $\mathsf{T}_m$ is a generalization.

It is trivial to see that a table that satisfies *k*-anonymity has a unique *k*-minimal generalization, which is itself. It is also easy to see that the necessary and sufficient condition for a table T to satisfy *k*-anonymity is that the cardinality of the table must be least *k*, as stated the following theorem. The requirement of the maximal elements of each $DGH_i$ to be a singleton ensures the sufficiency of the condition.

**Theorem 2**

Let T be a table and *k* be a natural number. If $|T| \geq k$, then there exists at least a *k*-minimal generalization for T. If $|T| < k$ there is no *k*-minimal generalization for T.

### 5.3.1 Distance vectors and generalization strategies

I introduce a distance vector metric with Euclidean properties that measures distances between tuples and between tables based on the number of generalizations or on the length of the functional generalization sequence required to have the tuples or tables share the same generalized values.

## Definition. Distance vector

Given $DGH_{Ali}$, with $f_h : h=0,\ldots,p$, where $i=1,\ldots,n$, and tables $T_l(A_{l1},\ldots,A_{ln})$ and $T_m(A_{m1},\ldots,A_{mn})$ such that $T_l \leq T_m$, the distance vector of $T_l$ to $T_m$ is the vector $DV_{l,m} = [d_1,\ldots,d_n]$ where each $d_i$ is the length of the unique path between $A_{lh}$,which is $A_{li}$ in $DGH_{Ali}$, and $A_{mh}$, which is $A_{mz}$ in $DGH_{Ali}$ or simply $m_h$-$l_h$.

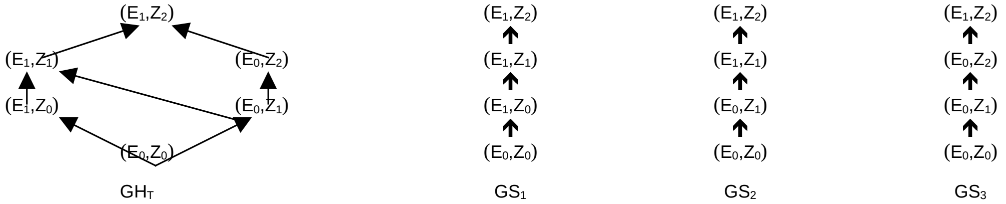


Figure 30 Generalization hierarchy $GH_T$ and strategies for T = <$E_0$,$Z_0$>

Intuitively the distance vector captures how many generalizations table $T_m$ is from Table $T_j$ for each attribute. To illustrate, consider private PT and its generalized tables illustrated in **Figure 27**. The distance vectors between PT and its different generalizations are the vectors appearing as subscripts to GT for each table.

The relationship between distance vectors and minimal generalizations, which is the basis of the correctness of my approach, is stated by the following theorem.

**Theorem 3**

Given tables $T_l$ and $T_m$ such that $T_l \leq T_m$ and $T_m$ satisfies *k*-anonymity. $T_m$ is *k*-minimal $\Leftrightarrow$ there does not exist a $T_z$ such that $T_z \neq T_l$, $T_z \neq T_m$ and $T_l \leq T_z$ and $T_z \leq T_m$, $T_z$ satisfies *k*-anonymity, and $DV_{l,z} \leq DV_{l,m}$.

Intuitively, the minimal generalizations of table $T_l$ are exactly those tables $T_j$ satisfying *k*-anonymity with minimal distance vectors $DV_{l,m}$. For instance, with reference to the generalized tables illustrated in **Figure 27**, I have already noticed how, for *k*=3, $GT_{[1,1]}$ cannot be minimal because $GT_{[0,1]}$ and $GT_{[1,0]}$ also satisfy *k*-anonymity. Recall that the subscript indicates the distance vector of the generalized table GT from PT.

Given $DGH_{Ai}$, with $f_{Aih} : h=0,\ldots,p_{Ai}$, where $i=1,\ldots,n$, and table $T(A_1,\ldots,A_n)$, the set of all possible generalizations of T comprise a generalization hierarchy, $GH_T = DGH_{A1} \times \ldots \times DGH_{An}$, assuming the Cartesian product is ordered by imposing coordinate-wise order [74]. $GH_T$ defines a lattice whose minimal element is T. For instance, **Figure 30** illustrates the generalization hierarchy $GH_{(E0,Z0)}$ where the domain generalization hierarchies of $E_0$ and $Z_0$ are as illustrated in **Figure 26**.

The generalization hierarchy of table T defines different ways in which T can be generalized. In particular each path from T to the unique maximal element of $GH_T$ in the graph describing $GH_T$ defines a possible alternative path they can be followed in the generalization process. I refer to the set of nodes in each such path together with the generalization relationships between them as generalization strategy for $GH_T$. The different generalization strategies for $GH_{(E0,Z0)}$ are illustrated in **Figure 30**. The number of different possible strategies for a generalization hierarchy is stated by the following theorem.

**Theorem 4**

Given $DGH_{Ai}$, with $f_{Aih} : h=0,\ldots,p_{Ai}$, where $i=1,\ldots,n$, and table $T(A_1,\ldots,A_n)$, the number of different generalization strategies for T is:

$$\frac{\left(\sum_{i=1}^{n} p_{Ai}\right)!}{\prod_{i=1}^{n} p_{Ai}!}$$

where each $p_{Ai}$ is the length of the path from $A_i$ to the maximal element in $\mathsf{DGH}_{\mathsf{Ai}}$.

For each strategy a minimal local generalization can be defined as the table satisfying *k*-anonymity, with sequence of domains DT*'* belonging to the strategy such that there are no other tables satisfying *k*-anonymity with sequence of domains DT" in the strategy and such that DT" ≤ DT'. This says the strategy is a total order and the minimal local generalization is always unique. The following theorem states the correspondence between *k*-minimal generalization and the local minimal generalization with respect to a strategy.

**Theorem 5**

Let T($A_1,...,A_n$) be a table to be generalized and let $\mathsf{GH}_\mathsf{T}$ be a generalization hierarchy for T. Every *k*-minimal generalization of T is a local minimal generalization for some strategy of $\mathsf{GH}_\mathsf{T}$.

The converse is not true; a local minimal generalization with respect to a strategy may not correspond to a *k*-minimal generalization. For instance, consider the table PT and its generalized tables illustrated in **Figure 27**, whose minimal results have been discussed in a previous example. For $k = 3$ the minimal local generalizations are: $\mathsf{GT}_{[1,0]}$ for strategy 1, $\mathsf{GT}_{[1,1]}$ for strategy 2 and $\mathsf{GT}_{[0,2]}$ for strategy 3. However, as I have shown in a previous example, $\mathsf{GT}_{[1,1]}$ is not *k*-minimal for $k = 3$. For $k = 2$ the minimal local generalizations are: $\mathsf{GT}_{[1,0]}$ for strategy 1 and $\mathsf{GT}_{[0,1]}$ for strategies 2 and 3. Directly from Theorem 5, a table has at most as many generalizations as the number of generalization strategies of its generalization hierarchy. The number of *k*-minimal generalizations can be smaller if the generalized table, locally minimal with respect to a strategy, is a generalization of a table locally minimal to another strategy ($\mathsf{GT}_{[1,1]}$ for $k = 3$ in the example above), or if different strategies have the same local minimal generalization ($\mathsf{GT}_{[0,1]}$ for $k = 2$ in the example above).

## 5.4 Minimal distortion of a table

When different minimal generalizations exist, preference criteria can be applied to choose a preferred solution among them. For example, tables which generalize (or not) specific attributes, or which return the highest number of distinct tuples can be preferred. For instance, for a *k*-anonymity requirement with $k = 2$, $\mathsf{GT}_{[1,0]}$ and $\mathsf{GT}_{[0,1]}$ are both minimal, as shown in **Figure 27**, but $\mathsf{GT}_{[0,1]}$ may be preferred because it contains a larger number of distinct tuples.

A natural measure for preferring one minimal generalization over another is based on selecting the minimal generalization whose information is least distorted. The application of any disclosure limitation technique [75] to a table T results in a table T' that has less information than T, and is therefore less pure than T; I say T' is a distorted version of T. In order to define the information loss precisely and specifically to the disclosure limitation techniques employed, I define an information theoretic metric that reports the amount of distortion of a table caused by generalization and suppression. While entropy is the classical measure commonly used in information theory to characterize the purity of data [76], and while information loss can therefore be simply expressed as the expected increase in entropy resulting from the application of a disclosure limitation technique, a metric based on the semantics of particular disclosure limitation techniques can be shown to be more discriminating than the direct comparison of the encoding lengths of the values stored in the table.

I can measure the distortion in a cell of the generalized table RT by computing the ratio of the domain of the value found within the cell to the height of the attribute's domain generalization hierarchy. The sum of the distortions found in each cell of the table RT provides an overall measure of the distortion of the table. The definition below defines the precision of a generalized table RT to be one minus the sum of the distortions found in the cells of the table (normalized by the total number of cells).

## Definition. precision metric

Let $\mathsf{PT}(A_1,...,A_{Na})$ be a table, $t_{Pj} \in \mathsf{PT}$, $\mathsf{RT}(A_1,...,A_{Na})$ be a generalization of PT, $t_{Pj} \in \mathsf{PT}$, each $\mathsf{DGH}_\mathsf{A}$ be the domain generalization hierarchy for attribute *A*, and $f_i$'s be generalizations on *A*. The precision of RT, written *Prec*(RT), based on generalization and suppression is:

$$Prec(\mathsf{RT}) = 1 - \frac{\sum_{i=1}^{N_A} \sum_{j=1}^{N} \frac{h}{\left|\mathsf{DGH}_{Ai}\right|}}{\left|\mathsf{RT}\right| \bullet \left|N_A\right|} \quad where\ f_1\left(\ldots f_h\left(t_{Pj}\left[A_i\right]\right)\ldots\right) = t_{Rj}\left[A_i\right]$$

**Example.**

Consider the trivial case where PT = RT. I that case, each value found within RT is in the ground domain of its attribute's domain generalization hierarchy and so each $h_{ij}$= 0; therefore, *Prec*(RT) = 1. Conversely, consider the trivial case where each value in each cell of RT is suppressed –i.e., the maximal element found in its attribute's domain generalization hierarchy. In that case, each $h_{ij}=|\text{DGH}_{\text{Ai}}|$; and so, *Prec*(RT) = 0.

**Example.**

Using the domain generalization hierarchies found in Figure 28 and Figure 29, I can compute the precision of the generalizations of the table labeled PT with respect to the quasi-identifier {*Ethnicity*, *ZIP*} that are found in **Figure 27**. The *Prec*(PT) = 1 because all values in this table are in their ground domains. *Prec*($\text{GT}_{[1,0]}$) = 0.75, the *Prec*($\text{GT}_{[1,1]}$) = 0.58, *Prec*($\text{GT}_{[0,2]}$) = 0.67, and *Prec*($\text{GT}_{[0,1]}$) = 0.83. Each of these generalizations satisfy *k*-anonymity for *k*=2, but $\text{GT}_{[0,1]}$ does so with minimal distortion.

As was shown in the previous example, there is inherent bias within *Prec* based on the height of the domain generalization hierarchies associated with the attributes of the table. Primarily, generalizations based on attributes with taller domain generalization hierarchies maintain precision better than generalizations based on attributes with shorter domain generalization hierarchies. For example, from **Figure 27**, $\text{GT}_{[1,0]}$ and $\text{GT}_{[0,1]}$ each generalize values up one level of an attribute's domain generalization hierarchy. But, from Figure 28 and Figure 29, $|\text{DGH}_{\text{Ethnicity}}|$ = 2 and $|\text{DGH}_{\text{ZIP}}|$ = 3 and so, *Prec*($\text{GT}_{[0,1]}$) > *Prec*($\text{GT}_{[1,0]}$).

**Requirement on domain generalization hierarchies**

For the semantics of the precision metric to be most accurate, domain generalization hierarchies used within the computation must be streamlined to contain no unnecessary or unattainable domains. Otherwise, the height of the domain generalization hierarchy will be arbitrarily increased and the precision metric cannot reach 0 with respect to the attribute. For example, if suppression is fixed atop the domain generalization hierarchy, then it should be removed from the precision analysis in cases where suppression cannot be achieved.

Of course the usefulness of a generalized table is specific to the application to which the data will be put [77]. Therefore, determining which minimal generalization is most useful relies on user-specific preferences. These preferences can be provided as: (1) weights incorporated in the weighted precision metric defined below; and, (2) a selection process for selecting among a set of minimal generalizations all of which have the same weighted precision.

## Definition. weighted precision metric

Let PT($A_1$,...,$A_{Na}$) be a table, $t_{Pj} \in$ PT, RT($A_1$,...,$A_{Na}$) be a generalization of PT, $t_{Pj} \in$ PT, each $\text{DGH}_A$ be the domain generalization hierarchy for attribute $A$, $f_i$'s be generalizations on $A$, and $W$ be the set of weights to specify preference where $w_{ij} \in W$ is a weight assigned to $t_{Pj}[A_i]$ such that $0 \le Prec_W(\text{RT}) \le 1$. The precision of RT, written $Prec_W(\text{RT})$, based on generalization and suppression is therefore:

$$Prec_W(\text{RT}) = 1 - \frac{\sum_{i=1}^{N_A} \sum_{j=1}^{N} \left( \frac{h}{|\text{DGH}_{Ai}|} \bullet w_{ij} \right)}{|\text{RT}| \bullet |N_A|}$$

$$where\ f_1(\ldots f_h(t_{Pj}[A_i])\ldots) = t_{Rj}[A_i]\ and\ 0 \le Prec_W(\text{RT}) \le 1$$

**Example.**

Recall in a previous example involving generalization enforced at the attribute level, both $\text{GT}_{[0,1]}$ and $\text{GT}_{[1,0]}$ in **Figure 27** were found to be minimal generalizations of PT [78]. A preference among minimal generalizations can be based on merely summing the level of generalization of each attribute with respect to the heights of the domain generalization hierarchies for those attributes. One level of generalization corresponds to the values within the table being associated with the next domain up the domain generalization hierarchy. In these cases preference is based on the minimal generalizations having the smallest sum of values found in the corresponding distance vectors. The weight $w_{ij}$ in the weighted precision metric in this case is:

$$w_{ij} = \frac{|\text{DGH}_{Ai}| \bullet N_A}{\sum_{k=1}^{N_A} |\text{DGH}_{Ak}|}$$

**Equation 3**

This simplifies $Prec_W$ to:

$$Prec_W(\mathsf{RT}) = 1 - \frac{\sum_{i=1}^{N_A} h_i}{\sum_{k=1}^{N_A} |\mathsf{DGH}_{Ak}|}$$

The weighted precision for the generalizations in **Figure 27** using the weight in Equation 3 is: $Prec_W(\mathsf{GT}_{[1,0]})$=0.8, $Prec_W(\mathsf{GT}_{[1,1]})$=0.6, $Prec_W(\mathsf{GT}_{[0,2]})$=0.6, and $Prec_W(\mathsf{GT}_{[0,1]})$=0.8. In this scheme, the minimal generalizations $\mathsf{GT}_{[0,1]}$ and $\mathsf{GT}_{[1,0]}$ are of equal preference and minimal distortion.

Notice that the regular precision metric *Prec* is a special case of the weighted precision metric $Prec_W$, where $\forall w_{ij} \in W$, $w_{ij} = 1$. In that case, $Prec(\mathsf{T}) = Prec_W(\mathsf{T})$. As was shown in the earlier example, not all minimal generalizations are equally distorted and preference can be based on the minimal generalization having the most precision. This concept is captured by the following definition of *k*-minimal distortion.

## Definition. *k*-minimal distortion

Let $\mathsf{T}_l(A_1,\ldots,A_n)$ and $\mathsf{T}_m(A_1,\ldots,A_n)$ be two tables such that $\mathsf{T}_l[\mathsf{QI}_\mathsf{T}] \leq \mathsf{T}_m[\mathsf{QI}_\mathsf{T}]$, where $\mathsf{QI}_\mathsf{T}=\{A_i,\ldots,A_j\}$ is the quasi-identifier associated with the tables and $\{A_i,\ldots,A_j\} \subseteq \{A_1,\ldots,A_n\}$ and $\forall x=i,\ldots,j$, $\mathsf{DGH}_{Ax}$ are domain generalization hierarchies for $\mathsf{QI}_\mathsf{T}$. $\mathsf{T}_m$ is said to be a minimal distortion of a table $\mathsf{T}_l$ with respect to a *k* anonymity requirement over $\mathsf{QI}_\mathsf{T}$ if and only if:

1. $\mathsf{T}_m$ satisfies the *k*-anonymity requirement with respect to $\mathsf{QI}_\mathsf{T}$
2. $\forall \mathsf{T}_z$: $Prec(\mathsf{T}_l) \geq Prec(\mathsf{T}_z)$, $Prec(\mathsf{T}_z) \geq Prec(\mathsf{T}_m)$, $\mathsf{T}_z$ satisfies the *k*-anonymity requirement with respect to $\mathsf{QI}_\mathsf{T} \Rightarrow \mathsf{T}_z[A_1,\ldots,A_n] = \mathsf{T}_m[A_1,\ldots,A_n]$.

**Example.**

**Figure 27** shows examples of generalizations of the table labeled PT with respect to the quasi-identifier {*Ethnicity*, *ZIP*}. Of these, only $\mathsf{GT}_{[0,1]}$ is a *k*-minimal distortion.

A *k*-minimal distortion is based on the precision metric *Prec*. Domain generalization with different heights can provide different *Prec* measures for the same table. So a *k*-minimal distortion is specific to a table, a quasi-identifier and a set of domain generalization hierarchies for the attributes of the quasi-identifier.

Also, the definition of *k*-minimal distortion can be modified to use the weighted precision metric $Prec_W$ rather than *Prec*. I term this result a weighted *k*-minimal distortion.

It is trivial to see that a table that satisfies *k*-anonymity has a unique *k*-minimal distortion, which is itself. It is also easy to see that a generalized table RT that is a *k*-minimal distortion of table PT is also a *k*-minimal generalization of PT, as stated in the following theorem.

**Theorem 6**

Given tables $T_l$ and $T_m$ such that $T_l \leq T_m$ and $T_m$ satisfies *k*-anonymity. $T_m$ is a *k*-minimal distortion of $T_l \Rightarrow T_m$ is *k*-minimal generalization of $T_l$.

## 5.5 An algorithm for determining a minimal generalization with minimal distortion

Figure 31 presents an algorithm, called MinGen, which, given a table **PT**($A_x,\ldots,A_y$), a quasi-identifier QI={$A_1,\ldots,A_n$}, where $\{A_1,\ldots,A_n\} \subseteq \{A_x,\ldots,A_y\}$, a *k*-anonymity constraint, domain generalization hierarchies $DGH_{Ai}$, produces a table **MGT** which is a *k*-minimal distortion of **PT**[*QI*]. It assumes that $k < |$**PT**$|$, which is necessary and sufficient condition for the existence of a minimal generalized table (see Theorem 2). The MinGen algorithm ties together the formal methods presented in this chapter and provides a model against which real-world systems will be compared in subsequent chapters.

```
Preferred Minimal Generalization (MinGen) Algorithm
Input:     Private Table PT; quasi-identifier QI = ($A_1$, …, $A_n$), k-anonymity constraint k;
           domain generalization hierarchies $DGH_{A_i}$, where i=1,…,n,
           and preference specifications a preferred() function.
Output:    MGT containing a minimal distortion of PT[QI] with respect to k-anonymity chosen
           according to the preference specifications
Assumes:   |PT |≥ k
Method:
3.  if PT[QI] satisfies k-anonymity requirement with respect to k then do
    3.1 MGT ← { PT }   // PT is the solution
4.  else do
    4.1 allgen ← {$T_i$ : $T_i$ is a generalization of PT over QI}
    4.2 protected ← {$T_i$ : $T_i \in$ allgen ∧ $T_i$ satisfies k-anonymity requirement of k}
    4.3 MGT ← {$T_i$ : $T_i \in$ protected ∧ there does not exist $T_z \in$ protected such that Prec($T_z$) > Prec($T_i$) }
    4.4 MGT ← preferred(MGT)  // select the preferred solution
5.  return MGT.
```

**Figure 31 Preferred MinGen Algorithm**

There are few steps in the MinGen algorithm. Step 1 determines if the original table, named PT, itself satisfies the *k*-anonymity requirement; and if so, it is the *k*-minimal distortion. Step 2 is the core of the algorithm executed in all other cases. Sub-step 2.1 stores the set of all possible generalizations of PT over the quasi-identifier QI in *allgens*. Recognizing that some of the generalizations in *allgens* satisfy the *k*-anonymity requirement and others do no, sub-step 2.2 stores those generalizations in *allgens* that do satisfy the *k*-anonymity requirement in *protected*. Sub-step 2.3 filters out those generalizations from *protected* that are not minimally distorted with respect to *Prec* and stores the resulting generalizations in *MGT*. Notice that $\forall GT_1, GT_2 \in MGT$, $Prec(GT_1) = Prec(GT_2)$. That is, after sub-step 2.3, *MGT* is the set of all *k*-minimal distortions of PT. It is guaranteed that $|MGT| \geq 1$. The function *preferred()* in sub-step 2.4 selects a single generalization from *MGT* based on user-defined specifications.

The algorithm is straightforward, so its correctness relies on the definitions of generalization [79], the *k*-anonymity requirement [80], and *Prec* [81]. It can be proved that a generalization of a table T over a quasi-identifier QI, that satisfies a given *k*-anonymity requirement, and has the least amount of distortion of all possible generalizations of T over QI, is a *k*-minimal distortion of T over QI with respect to *Prec*. From Theorem 6, the solution is also a *k*-minimal generalization of T over QI.

The MinGen algorithm assumes there are at least *k* tuples in PT. The maximal element requirement atop each domain generalization hierarchy assures |*protected*| ≥ 1 in all cases. For example, *protected* always includes the table consisting of tuples which are all the same and indistinguishable, where each value within each tuple is the maximal generalized element for its attribute.

With respect to complexity, MinGen makes no claim to be efficient. The |*allgens*| was expressed in Equation 1, if generalization is enforced at the attribute level, and in Equation 2, if generalization is enforced at the cell level. In both cases, the computational cost is tremendous, making an exhaustive search of all possible generalizations impractical on even the most modest of tables.

Care must be taken that the domain generalization hierarchies used by MinGen contain domains that are attainable by MinGen; otherwise, the height of the domain generalization hierarchy is inflated and so, *Prec* can never be 0. For example, the domain generalization hierarchies based on the depictions in Figure 32 include only those domains that can be attained by the MinGen algorithm. The depictions in Figure 33 include an additional domain atop each hierarchy, where the additional domain contains the suppressed value for the attribute. However, the MinGen algorithm would never provide a solution that contained any suppressed values given those hierarchies. Therefore, the hierarchies in Figure 33 when used by MinGen fail the requirement that *Prec* can achieve 0 [82]; so, the hierarchies in Figure 32 should be used with MinGen.

In sub-step 2.4 of the MinGen algorithm, in cases where |*MGT*| > 1, each table in *MGT* is a solution, but the *preferred()* function can return only one table as a solution. This single solution requirement is a necessary condition because the chosen solution is then considered to become part of the join of external information against which subsequent linking and matching must be protected. This places additional constraints on the subsequent release of any other tables in *MGT* and of other generalizations of the privately held information. Here are three related attacks and their solutions.

### 5.5.1 Unsorted matching attack against *k*-anonymity

This attack is based on the order in which tuples appear in the released table. While I have maintained the use of a relational model in this discussion, and so the order of tuples cannot be assumed, in real-world use this is often a problem. It can be corrected of course, by randomly sorting the tuples of the solution. Otherwise, the release of a related table can leak sensitive information.

**Example.**

Using a weighted precision metric with the weight described in Equation 3, $GT_{[0,1]}$ and $GT_{[1,0]}$ in **Figure 27** are both *k*-minimal distortions of PT, where *k*=2. If $GT_{[0,1]}$ is released and a subsequent release of $GT_{[1,0]}$ is then performed, but where the position of the tuples in each table correspond to the same tuple in PT, then direct matching of tuples across the tables based on tuple position within the tables reveals sensitive information. On the other hand, if the positions of the tuples within each table are randomly determined, both tables can be released.

### 5.5.2 Complementary release attack against *k*-anonymity

In the previous example, all the attributes in the generalized tables were in the quasi-identifier. That is typically not the case. It is more common that the attributes that constitute the quasi-identifier are themselves a subset of the attributes released. As a result, when a *k*-minimal solution, which I will call table T is released, it should be considered as joining other external information. Therefore, subsequent releases of generalizations of the same privately held information must consider all of the released attributes of T a quasi-identifier to prohibit linking on T, unless of course, subsequent releases are themselves generalizations of T.

**Example.**

Consider the private table PT in Figure 34. The tables GT1, GT2 and GT3 in Figure 35 were identified by MinGen (after step 2.3) as *k*-minimal distortions of PT, where *k*=2, the quasi-identifier QI={*Race*, *BirthDate*, *Gender*, *ZIP*} and the domain generalization hierarchies are based on the depictions in Figure 32. Suppose table GT1 is released as the preferred *k*-minimal solution. If subsequently GT3 is also released, then the *k*-anonymity protection will no longer hold, even if the tuple positions are randomly determined in both tables. Linking GT1 and GT3 on {*Problem*} reveals the table LT shown in Figure 36. Notice how [white, 1964, male, 02138] and [white, 1965, female, 02139] are unique in LT and so, LT does not satisfy the *k*-anonymity requirement enforced by GT1 and GT3. This problem would not exist if GT3 used the quasi=identifier QI $\cup$ {*Problem*} or if a generalization of GT1 had been released instead of GT3.

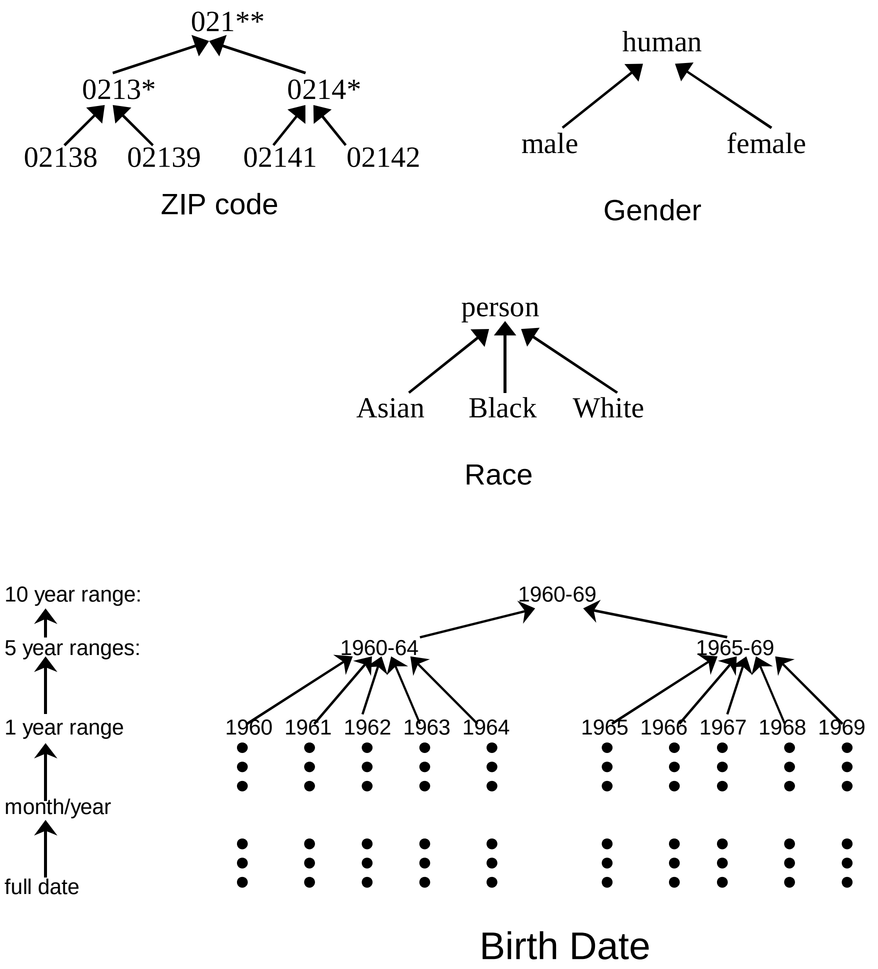


**Figure 32 Value generalization hierarchies for {*ZIP*, *Gender*, *Race*, *BirthDate*}**

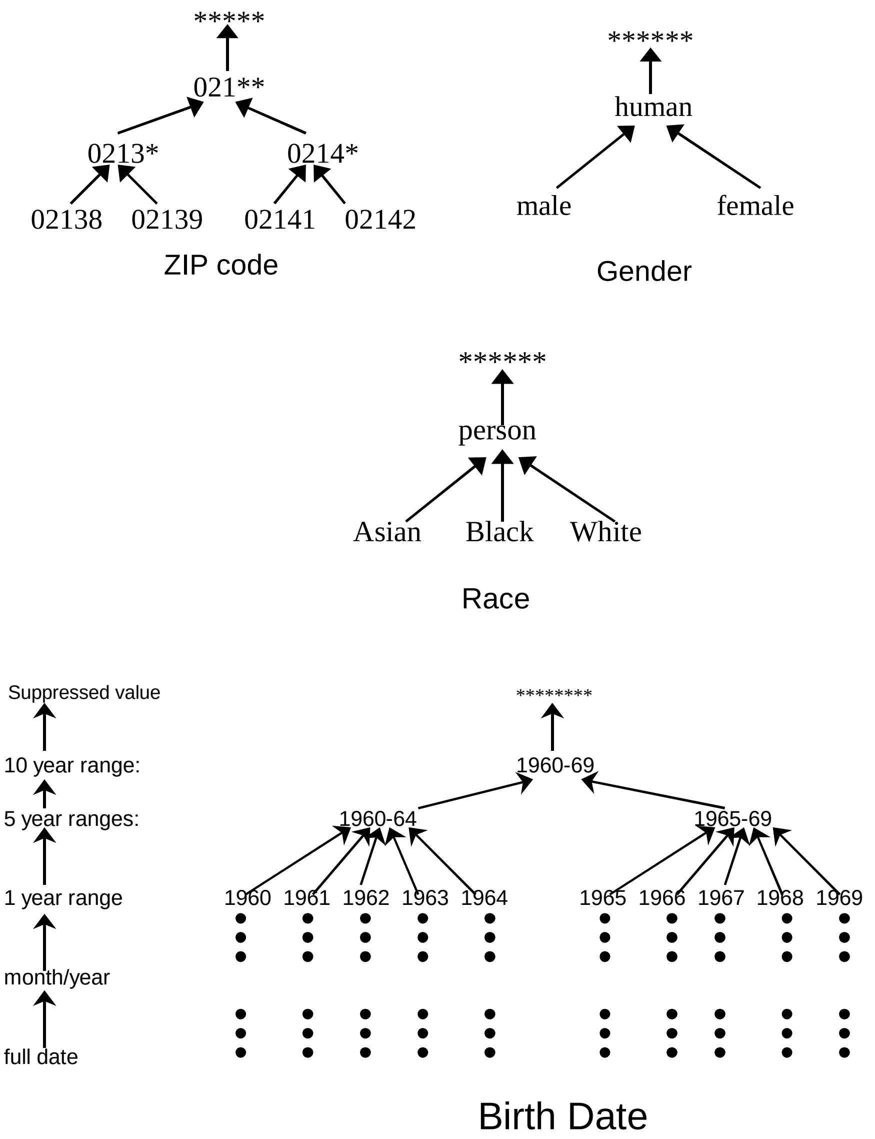


**Figure 33 Value generalization hierarchies for {*ZIP, Gender, Race, BirthDate*} with suppression**

| id | Race | BirthDate | Gender | ZIP | Problem |
|---|---|---|---|---|---|
| t1 | black | 9/1965 | male | 02141 | short of breath |
| t2 | black | 2/1965 | male | 02141 | chest pain |
| t3 | black | 10/1965 | female | 02138 | painful eye |
| t4 | black | 8/1965 | female | 02138 | wheezing |
| t5 | black | 11/1964 | female | 02138 | obesity |
| t6 | black | 12/1964 | female | 02138 | chest pain |
| t7 | white | 10/1964 | male | 02138 | short of breath |
| t8 | white | 3/1965 | female | 02139 | hypertension |
| t9 | white | 8/1964 | male | 02139 | obesity |
| t10 | white | 5/1964 | male | 02139 | fever |
| t11 | white | 2/1967 | male | 02138 | vomiting |
| t12 | white | 3/1967 | male | 02138 | back pain |

**Figure 34 Private Table PT**

| Race | BirthDate | Gender | ZIP | Problem |
|---|---|---|---|---|
| black | 1965 | male | 02141 | short of breath |
| black | 1965 | male | 02141 | chest pain |
| person | 1965 | female | 0213* | painful eye |
| person | 1965 | female | 0213* | wheezing |
| black | 1964 | female | 02138 | obesity |
| black | 1964 | female | 02138 | chest pain |
| white | 1964 | male | 0213* | short of breath |
| person | 1965 | female | 0213* | hypertension |
| white | 1964 | male | 0213* | obesity |
| white | 1964 | male | 0213* | fever |
| white | 1967 | male | 02138 | vomiting |
| white | 1967 | male | 02138 | back pain |

GT1

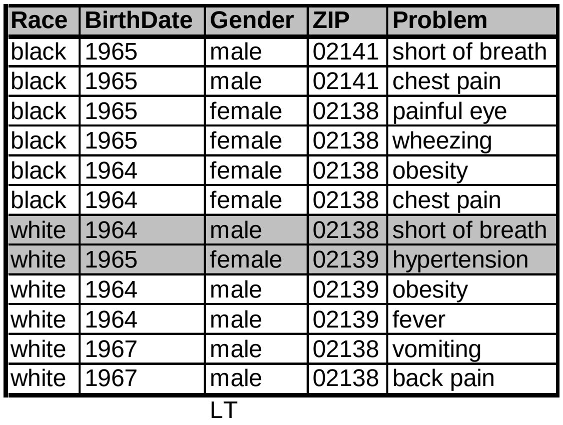
| Race | BirthDate | Gender | ZIP | Problem |
|---|---|---|---|---|
| black | 1965 | male | 02141 | short of breath |
| black | 1965 | male | 02141 | chest pain |
| person | 1965 | female | 0213* | painful eye |
| person | 1965 | female | 0213* | wheezing |
| black | 1964 | female | 02138 | obesity |
| black | 1964 | female | 02138 | chest pain |
| white | 1960-69 | male | 02138 | short of breath |
| person | 1965 | female | 0213* | hypertension |
| white | 1964 | male | 02139 | obesity |
| white | 1964 | male | 02139 | fever |
| white | 1960-69 | male | 02138 | vomiting |
| white | 1960-69 | male | 02138 | back pain |

GT2

| Race | BirthDate | Gender | ZIP | Problem |
|---|---|---|---|---|
| black | 1965 | male | 02141 | short of breath |
| black | 1965 | male | 02141 | chest pain |
| black | 1965 | female | 02138 | painful eye |
| black | 1965 | female | 02138 | wheezing |
| black | 1964 | female | 02138 | obesity |
| black | 1964 | female | 02138 | chest pain |
| white | 1960-69 | male | 02138 | short of breath |
| white | 1960-69 | human | 02139 | hypertension |
| white | 1960-69 | human | 02139 | obesity |
| white | 1960-69 | human | 02139 | fever |
| white | 1960-69 | male | 02138 | vomiting |
| white | 1960-69 | male | 02138 | back pain |

GT3

**Figure 35 *k*-minimal distortions for PT in Figure 34 where *k*=2**



| Race | BirthDate | Gender | ZIP | Problem |
|---|---|---|---|---|
| black | 1965 | male | 02141 | short of breath |
| black | 1965 | male | 02141 | chest pain |
| black | 1965 | female | 02138 | painful eye |
| black | 1965 | female | 02138 | wheezing |
| black | 1964 | female | 02138 | obesity |
| black | 1964 | female | 02138 | chest pain |
| white | 1964 | male | 02138 | short of breath |
| white | 1965 | female | 02139 | hypertension |
| white | 1964 | male | 02139 | obesity |
| white | 1964 | male | 02139 | fever |
| white | 1967 | male | 02138 | vomiting |
| white | 1967 | male | 02138 | back pain |

LT

**Figure 36 Table resulting from linking GT1 and GT3 in Figure 35**

### 5.5.3 Temporal attack against *k*-anonymity

Data collections are dynamic. Tuples are added, changed, and removed constantly. As a result, releases of generalized data over time can be subject to a temporal inference attack. Let table $T_0$ be the original privately held table at time *t*=0. Assume a *k*-minimal solution of $T_0$, which I will call table $RT_0$, is released. At time t, assume additional tuples were added to the privately held table T0, so it comes $T_t$. Let $RT_t$ be a *k*-minimal solution of $T_t$ that is released at time *t*. Because there is no requirement that $RT_t$ respect the distortions of $RT_0$, linking the tables $RT_0$ and $RT_t$ may reveal sensitive information and thereby compromise *k*-anonymity protection. As was the case in the previous example, to combat this problem, $RT_0$ should be considered as joining other external information. Therefore, either all of the attributes of $RT_0$ would be considered a quasi-identifier for subsequent releases, or subsequent releases themselves would be generalizations of $RT_0$.

**Example.**

At time $t_0$, assume the privately held information is PT in Figure 34. As stated earlier, GT1, GT2 and GT3 in Figure 35 are *k*-minimal distortions of PT over the quasi-identifier QI={*Race*, *BirthDate*, *Gender*, *ZIP*} where *k*=2. Assume GT1 is released. At a later time $t_1$, PT becomes $PT_{t1}$, which is PT $\cup$ {[black, 9/7/65, male, 02139, headache], [black, 11/4/65, male, 02139, rash]}. MinGen executes on $PT_{t1}$ as it has on PT and returns a *k*-minimal distortion, which I will call $GT_{t1}$. Assume this table contains GT3 in Figure 35; specifically, $GT_{t1}$ = GT3 $\cup$ {[black, 1965, male, 02139, headache], [black, 1965, male, 02139, rash]}. As was shown in an earlier example, GT1 and GT3 can be linked on {*Problem*} to reveal unique tuples over QI. Likewise, GT1 and $GT_{t1}$ can be linked to reveal the same unique tuples. One way to combat this problem is run MinGen on GT1 ∪ ($PT_{t1}$ – PT), making the result a generalization of GT1. In that case, a result could be GT1 ∪ {**[**`black`, `1965`, `male`, `02139`, `headache`**]**, **[**`black`, `1965`, `male`, `02139`, `rash`**]**}, which does not compromise the distorted values in GT1.

### 5.5.4 MinGen as an anonymous data system

MinGen uses the generalization and suppression as disclosure limitation techniques. Below is a description of the framework in which MinGen operates.

S = {*subjects whose information is included in PT*}
P = *set of all people whose information could possibly be in PT*
PT = *privately held information about S*
*QI = set of attributes with replications in E*
U = *P*
RT = *MinGen*(PT)
E = set of publicly available information in today's society
G = set of standard communication methods.
*f = MinGen*

The system **A**(S, P, PT, *QI*, U, {RT}, E, G, *MinGen*) is an $\mathbf{ADS_{0.}}$

Informal proof.
If *QI* contains all attributes replicated in E, **A** adheres to *k*-map protection,
where *k* is enforced on RT. That is, for each value of *QI* released in RT,
there are at least *k* tuples having that value.
So, **A** is an $\mathbf{ADS_{0.}}$

The practical significance of releasing individualized data, such that linking of the data to other sources to re-identify individuals cannot be done, offers many benefits to our electronic society. This work provides an effective and optimal solution to this problem.

In the next chapters, I present four computational systems that attempt to maintain privacy while releasing electronic information. These systems are: (1) my Datafly II System, which generalizes and suppresses values in field-structured data sets [83]; (2) Statistics Netherlands' μ-Argus System, which is becoming a European standard for producing public-use data [84]; (3) my *k*-Similar algorithm, which produces optimal results in comparison to Datafly and μ-Argus [85]; and, (4) my Scrub System, which locates personally-identifying information in letters between doctors and notes written by clinicians [86]. The Datafly, μ-Argus and *k*-Similar systems primarily use generalization and suppression for disclosure limitation and provide protection by seeking to adhere to *k*-anonymity. As was shown in Equation 1 (on page 87) and Equation 2 (on page 87), the number of possible generalizations prohibits an exhaustive search, as was done by MinGen. As a result, these systems make approximations, which may not always yield optimal results. In the next chapters, I assess the anonymity protection provided by each of these systems in terms of whether each system is an $\mathbf{ADS_0}$ and compare the performance of each to MinGen. The presentation returns to an informal style.

## 5.6 Future work

1. The size of and conditions for *k* necessary to ensure *k*-anonymity must be further investigated. The Social Security Administration (SSA) releases public-use files based on national samples with small sampling fractions (usually less than 1 in 1,000); the tuples contain no geographic codes, or at most regional or size of place designators [87]. The SSA recognizes that data containing individuals with unique combinations of characteristics can be linked or matched with other data sources. So, the SSA's general rule is that any subset of the data that can be defined in terms of combinations of characteristics must contain at least 5 individuals. This condition for *k* includes a sampling fraction and no geographical specification. Current demand requires releasing all data with geographical specification. How does this change the size of *k*? Studies could be based on a cost of communication model, where the size of *k* is related to the cost of communicating with candidates to determine the correct identity of persons who are the subject of the data.

2. The quality of generalized data is best when the attributes most important to the recipient do not belong to any quasi-identifier. For public-use files this is acceptable, but determining the quality and usefulness in other settings must be further researched. Survey published results and determine which studies, if any, could have been achieved with sufficiently anonymous

data rather than identified data and which, if any, could not have used sufficiently anonymous data without skewing results or prohibiting them altogether. Candidate studies include epidemiological studies and surveys. Classify the results.

3. This chapter extended some of the foundational methods provided in the previous chapter. In particular, this chapter focused on one version of *k*-map protection, namely *k*-anonymity, and employed two disclosure limitation techniques, namely generalization and suppression. There are other protection models [88] and other techniques [89]. Select another protection model and/or other disclosure limitation techniques and extend the methods.

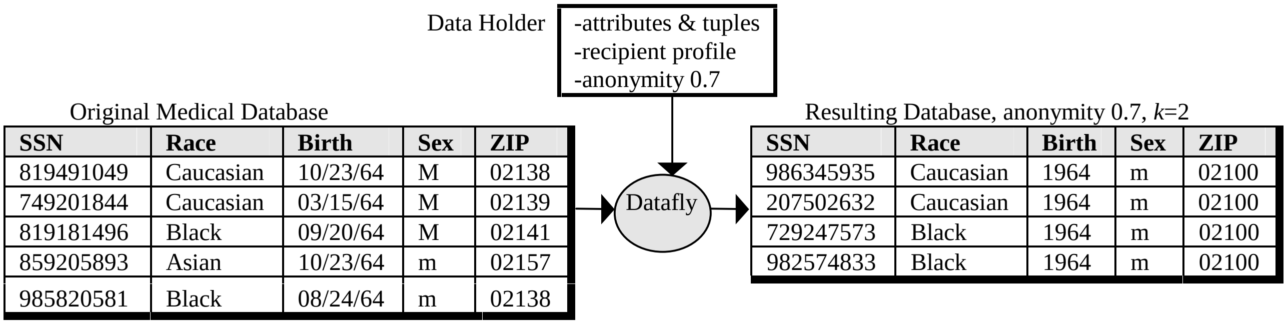
4. Disclosure limitation has been performed in different communities on different kinds of data –such as summary tables, geographical information systems, textual documents and even, DNA sequences. While the list of disclosure limitation techniques provided earlier (on page 60), crosses these boundaries, some techniques may work better with some kinds of data and uses than others. Perform an analysis to see which kinds of disclosure limitation techniques work best with which kinds of data and uses and why.

5. Weighted metrics were defined among the methods introduced in this chapter. Consider data in a given application area, such as hospital discharge data, and introduce strategies for how weights could be strategically applied to convey notions that some fields of information contain information more sensitive than others. Try out the proposed schemes and compare the semantics of the results to the non-weighted version.

# Chapter 6 Results: Datafly II

In this chapter, I present my Datafly and Datafly II Systems whose goal is to provide the most general information useful to the recipient. From now on, the term Datafly will refer to the Datafly II System unless otherwise noted. Datafly maintains anonymity in released data by automatically substituting, generalizing and suppressing information as appropriate. Decisions are made at the attribute and tuple level at the time of database access, so the approach can be incorporated into role-based security within an institution as well as in exporting schemes for data leaving an institution. The end result is a subset of the original database that provides minimal linking and matching of data because each tuple matches as many people as the data holder specifies.

## 6.1 Overview of the Datafly System



Original Medical Database

| SSN | Race | Birth | Sex | ZIP |
|---|---|---|---|---|
| 819491049 | Caucasian | 10/23/64 | M | 02138 |
| 749201844 | Caucasian | 03/15/64 | M | 02139 |
| 819181496 | Black | 09/20/64 | M | 02141 |
| 859205893 | Asian | 10/23/64 | m | 02157 |
| 985820581 | Black | 08/24/64 | m | 02138 |

Resulting Database, anonymity 0.7, *k*=2

| SSN | Race | Birth | Sex | ZIP |
|---|---|---|---|---|
| 986345935 | Caucasian | 1964 | m | 02100 |
| 207502632 | Caucasian | 1964 | m | 02100 |
| 729247573 | Black | 1964 | m | 02100 |
| 982574833 | Black | 1964 | m | 02100 |

**Figure 37. Data holder overview of the Datafly System**

Figure 37 provides an overview of the Datafly System from the data holder's perspective for generating a table for release. The original table is shown on the left. Input to the Datafly System is the original privately held table and some specifications provided by the data holder. Output is a table whose attributes and tuples correspond to the anonymity level specified by the data holder; in Figure 37 the anonymity level is noted as being 0.7. These terms and the process used by Datafly to generate a table for release are discussed in the following paragraphs.

Before any releases are generated, each attribute in the original table is tagged as using either an equivalence class substitution algorithm or a generalization routine when its associated values are to be released. If values of an attribute tagged as using equivalence class substitution are to be released, made-up alternatives replace values of the attribute in the released data. The Social Security number attribute labeled *SSN* provides an example in Figure 37 and a strong one-way hashing (encryption) algorithm is used.

Alternatively, if an attribute is tagged as using generalization, then an accompanying generalization hierarchy is assigned to the attribute; example hierarchies are shown in Figure 33 on page 101. The Datafly System iteratively computes increasingly less specific versions of the values for the attribute until eventually the desired anonymity level is attained. For example, the *birth date* attribute would first have the full month, day and year for each value. If further generalization were necessary, only the month and year would be used, and then only the year and so on, as the values get less and less specific, moving up the generalization hierarchy. The iterative process ends when there exists *k* tuples having the same values assigned across a group of attributes (or quasi-identifier); this is termed a *k* requirement and provides the basis for *k*-anonymity protection discussed earlier [90]. [Note in the earliest version of Datafly, *k* was enforced on each attribute individually and a complicated requirement was enforced across attributes; but in later versions which are named Datafly II, *k* is enforced across the quasi-identifier as described here.] In Figure 37 the quasi-identifier under consideration, because of the size of the database shown, is only {*Race*, *Birth*, *Sex*, *ZIP*} and *k*=2; therefore, in the released data, there are at least two tuples for each combination of {*Race*, *Birth*, *Sex*, *ZIP*} released.

To use the system, the data holder (1) declares specific attributes and tuples in the original private table as being eligible for release. The data holder also (2) groups a subset of the released attributes into one or more quasi-identifiers and provides (3) a number from 0 to 1 is assigned to each attribute eligible for release that identifies the likelihood each attribute within a quasi-identifier will be used for linking; a 0 value means not likely and a value of 1 means highly probable. I term such a list a *profile*. Finally, the data holder (4) specifies a minimum overall anonymity level that computes to a value of *k* and (5) a threshold (called *loss*) that determines the maximum number of tuples that can be suppressed, where *loss* must correspond to at least *k* tuples.

Datafly then produces the released table from the eligible attributes and tuples of the private table such that each value of a quasi-identifier in the released table appears in at least *k* tuples. The *k* requirement is accomplished by generalizing attributes within a quasi-identifier as needed and suppressing no more than *loss* tuples.

In Figure 37, notice how the record containing the Asian entry was removed; Social Security numbers were automatically replaced with made-up alternatives; birth dates were generalized to the year and ZIP codes to the first three digits. In the next two paragraphs I examine the overall anonymity level and its relationship to *k* and *loss*.

The overall anonymity level is a number between 0 and 1 that relates to the minimum *k* for each quasi-identifier. An anonymity level of 0 provides the original data and a level of 1 forces Datafly to produce the most general data possible given the profile of the recipient. All other values of the overall anonymity level between 0 and 1 determine the operational value for *k*. (The institution is responsible for mapping the anonymity level to particular values of *k*.) Information within each attribute is generalized as needed to attain the minimum *k* and *outliers,* which are extreme values not typical of the rest of the data, may be removed. Upon examination of the resulting data, every value assigned to each quasi-identifier will occur at least *k* times with the exception of one-to-one replacement values, as is the case with Social Security numbers.

| **Anonymity (*A*)** | ***k*** | **Birth Date** | **maxDrop%** |
|---|---|---|---|
| 1 | | | |
| --- .9 --- | 493 | 24 | 4% |
| --- .8 --- | 438 | 24 | 2% |
| --- .7 --- | 383 | 12 | 8% |
| --- .6 --- | 328 | 12 | 5% |
| --- .5 --- | 274 | 12 | 4% |
| --- .4 --- | 219 | 12 | 3% |
| --- .3 --- | 164 | 6 | 5% |
| --- .2 --- | 109 | 4 | 5% |
| --- .1 --- | 54 | 2 | 5% |
| 0 | | | |

**Figure 38. Anonymity generalizations for Cambridge voters' data with corresponding values of *k*.**

Figure 38 shows the relationship between *k* and selected anonymity levels (*A*) using the Cambridge voters' database [91]. As *A* increased, the minimum requirement for *k* increased, and in order to achieve the *k*-based requirement, values within an attribute in a quasi-identifier, for example, *Birth Date*, were re-coded in ranges of 2, 4, 6, 12 or 24 months, as shown. Outliers were excluded from the released data, and their corresponding percentages of *N* (where *N* is the number of tuples in the privately held table eligible for release) are noted. An anonymity level of 0.7, for example, required at least 383 occurrences of every value of the quasi-identifier. To accomplish this in only *Birth Date*, for example, required re-coding dates to reflect only the birth year. Even after generalizing over a 12 month window, the values of 8% of the voters still did not meet the requirement so these voters were dropped from the released data.

In addition to an overall anonymity level, the data holder also provides a profile of the needs of the person who is to receive the data by specifying for each attribute that is to be in the release whether the recipient could have or would use information external to the database that includes data within that attribute. That is, the data holder estimates on which attributes the recipient might link outside knowledge. Thus, each attribute has associated with it a profile value between 0 and 1, where 0 represents full trust of the recipient or no concern over the sensitivity of the information within the attribute, and 1 represents full distrust of the recipient or maximum concern over the sensitivity of the attribute's contents. Semantically related attributes that are sensitive to linking, with the exception of one-to-one replacement attributes, are treated as a single concatenated attribute (a quasi-identifier) that must meet the minimum *k* requirement, thereby thwarting linking attempts that use combinations of attributes. The role of these profiles is to help select which attribute within the quasi-identifier will be selected for generalization. If all attributes in the quasi-identifier have the same value, then the attribute having the greatest number of distinct values will be generalized.

Consider the profiles of a doctor caring for a patient, a clinical researcher studying risk factors for heart disease, and a health economist assessing the admitting patterns of physicians. Clearly, these profiles are all different. Their selection and specificity of attributes are different; their sources of outside information on which they could link are different; and their uses for the data are different. From publicly available birth certificates, driver license, and local census databases, the birth dates, ZIP codes and gender of individuals are commonly available along with their corresponding names and addresses; so these attributes could easily be used for re-identification. Depending on the recipient, other attributes may be even more useful. If the recipient is the patient's caretaker within the institution, the patient has agreed to release this information to the care-taker, so the profile for these attributes should be set to 0 to give the patient's caretaker full access to the original information.

When researchers and administrators make requests that require less specific information than that originally provided within sensitive attributes, the corresponding profile values should warrant a number as close to 1 as possible, but not so much so that the resulting generalizations provide useless data to the recipient. But researchers or administrators bound by contractual and legal constraints that prohibit their linking of the data are trusted, so if they make a request that includes sensitive attributes, the profile values would ensure that each sensitive attribute adheres only to the minimum *k* requirement.

The goal is to provide the most general data that are acceptably specific to the recipient. Since the profile values are set independently for each attribute, particular attributes that are important to the recipient can result in less generalization than other requested attributes in an attempt to maintain the usefulness of the data. A profile for data being released for public use, however, should be 1 for all sensitive attributes to ensure maximum protection. The purpose of the profiles are to quantify the specificity required in each attribute and to identify attributes that are candidates for linking; and in so doing, the profiles identify the associated risk to patient confidentiality for each release of data.

Using a pediatric medical record system [92] consisting of 300 patient records with 7617 visits and 285 attributes stored in over 12 relational database tables, I conducted test in which the Datafly System processed all queries to the database over a spectrum of recipient profiles and anonymity levels to show that all attributes in medical records can be meaningfully generalized as needed because any attribute can be a candidate for linking. Of course, which attributes are most important to protect depends on the recipient. Attention was paid primarily to attributes commonly exported to government agencies, researchers and consultants. Diagnosis codes have generalizations using the International Classification of Disease (ICD-9) hierarchy (or other useful semantic groupings). Geographic replacements for states or ZIP codes generalize to use regions and population size. Continuous variables, such as dollar amounts and clinical measurements, can be converted to discrete values. Replacement values must be based on meaningful subdivisions of values; and, replacement need only be done in cases where the attributes are candidates for linking.

In the real-world example mentioned earlier on page 50, the Group Insurance Commission in Massachusetts (GIC) collected patient-specific data with almost 100 attributes of information per physician visit for a population of more than 135,000 state employees, their families and retirees. In a public hearing, GIC reported giving a copy of the data to a researcher, who in turn stated that she did not need the full date of birth, just the birth year [93]. The average value of *k* based only on {*birth date*, *gender*} for that population is 3, but had the researcher received only {*year of birth*, *gender*}, the average value of *k* would have increased to 1125. Furnishing the most general information the recipient can use minimizes unnecessary risk to patient confidentiality.

## 6.2 Abstract of the Datafly algorithm

Here is a summary of the setting in which the core Datafly algorithm operates. The data holder provides an overall anonymity level (*A*), which is a value between 0 and 1. The data holder also provides a profile of the recipient by providing a linking likelihood ($P_f$) for each attribute that is also a value between 0 and 1. Based on these values an overall value for *k* is computed and quasi-identifier(s) are determined. For example, subsets of attributes where $P_f$=1 are treated as one concatenated attribute, or quasi-identifier, which must satisfy a *k*-anonymity requirement. Each attribute has a replacement algorithm that either uses equivalence class substitution, such as SSNs, or generalization based on a domain generalization hierarchy specific to that attribute. Datafly also has a special facility for cases involving multiple tuples attributable to the same person because the number of occurrences and other information contained in the tuples, such as relative dates, can combine to reveal sensitive information. For simplicity however, I will remove many of these finer features of the Datafly System from my analysis of the underlying algorithm, with no loss of overall characterization. I describe the core Datafly algorithm as working with a quasi-identifier and a *k*-anonymity requirement that is to be enforced on the quasi-identifier. For convenience, I consider all attributes of the quasi-identifier as having equal weights (specifically, $P_f$=1 for each attribute of the quasi-identifier), so they can be considered as not having weights at all; and, I address only generalizable attributes of the quasi-identifier in isolation.

**Core Datafly Algorithm**

**Input:** Private Table PT; quasi-identifier QI = ($A_1$, …, $A_n$), $k$-anonymity constraint $k$; domain generalization hierarchies $\text{DGH}_{Ai}$, where $i$=1,…,$n$ with accompanying functions $f_{Ai}$, and *loss*, which is a limit on the percentage of tuples that can be suppressed. PT[*id*] is the set of unique identifiers (key) for each tuple.

**Output:** MGT a generalization of PT[QI] that enforces $k$-anonymity

**Assumes:** |PT |≥ $k$, and *loss* * |PT| = $k$

**algorithm Datafly:**

```
// Construct a frequency list containing unique sequences of values across the quasi-identifier in PT,
// along with the number of occurrences of each sequence.
1.  let freq be an expandable and collapsible Vector with no elements initially.Each element is of the
    form (QI, frequency, SID), where $SID = \{id_i : \exists t[id] \in PT[id] \Rightarrow t[id]=id_i\}$; and, $frequency = |SID|$.
    Therefore, freq is also accessible as a table over (QI, frequency, SID).
2.  let $pos \leftarrow 0$, $total \leftarrow 0$
3.  while total ≠ |PT| do
    5.1 $freq[pos] \leftarrow (t[QI], occurs, SID)$
                where $t[QI] \in PT[QI]$, $(t[QI],\_\_, \_\_\_) \notin freq$; $occurs = |PT| - |PT[QI] – \{t[QI]\}|$;
                    and, $SID = \{id_i : \exists t[id] \in PT[id] \Rightarrow t[id]=id_i\}$
    5.2 $pos \leftarrow pos + 1$, $total \leftarrow total + occurs$
// Make a solution by generalizing the attribute with the most number of distinct values
// and suppressing no more than the allowed number of tuples.
6.  let $belowk \leftarrow 0$
7.  for $pos \leftarrow 1$ to |freq| do
    7.1 $(\_\_, count) \leftarrow freq[pos]$
    7.2 if $count < k$ then do
        7.2.1   $belowk \leftarrow belowk + count$
8.  if $belowk > k$ then do:            // Note. $loss * |PT| = k$
    8.1 freq ← generalize(freq)
    8.2 go to step 4
9.  else do
// assert: the number of tuples to suppress in freq is ≤loss * |PT|
    9.1 freq ← suppress(freq, belowk)
    9.2 MGT ← reconstruct(freq)
10. return MGT.
```

**Figure 39 Core Datafly algorithm**

```
Datafly generalize Algorithm
// This algorithm identifies the attribute within the quasi-identifier having the most number of distinct
// values in the tuples stored in freq and then generalizes those values in freq. Generalization is
// enforced at the attribute level, so all the values associated with an attribute are in the same domain.
1. let $max \leftarrow 0$
2. for each $a \in$ QI do:
        2.1 let $values \leftarrow \varnothing$
        2.2 for $pos \leftarrow 1$ to |freq| do:
                2.2.1   $(t, \_\_, \_\_) \leftarrow$ freq[$pos$]
                2.2.2   $values \leftarrow values \cup \{ t[a] \}$
        // assert: values contains set of values assigned to attribute a in the tuples of freq
        2.3 if $max < |values|$ then do:
                2.3.1   $max \leftarrow |values|$
                2.3.2   $attr \leftarrow a$
// assert: attr is the attribute of QI having the most number of distinct values (max) in the tuples of freq
3. let V be a frequency list of the same type as freq. V initially has no elements.
4. if $max = 1$ then do:
        4.1 halt on error   // |PT| < $k$
// generalize values assigned to attr
5. for $pos \leftarrow 1$ to |freq| do:
        5.1 $([v_{a1}, \ldots, v_{an}], count, sid) \leftarrow$ freq[$pos$]
        5.2 if $attr = a_1$ then do
                5.2.1   V ← VectorAdd(V, $[f_{attr}(v_{a1}),\ldots,v_{an}]$, $count$, $sid$)
        5.3 else if $attr = a_n$ then do:
                5.3.1   V ← VectorAdd(V, $[v_{a1},\ldots,f_{attr}(v_{an})]$, $count$, $sid$)
        5.4 else V ← VectorAdd(V, $[v_{a1},\ldots,f_{attr}(v_{attr}),\ldots,v_{an}]$, $count$, $sid$)
6. freq ← V
7. return freq
```

**Figure 40 generalize(), supporting method for core Datafly algorithm**

```
Datafly VectorAdd Algorithm
Input:      V, $t$, $occurs$, $sid$
Output:     Updates and returns V, a frequency list
// This method adds the tuples associated with (t,occurs,sid) to V avoiding duplication
algorithm VectorAdd:
1. for $pos \leftarrow 1$ to |V| do:
        1.1. let $(t_1, occurs_1, sid_1) \leftarrow$ V[$pos$]
        1.2. if $t_1 = t$ then do:
                1.2.1.  V[$pos$] $\leftarrow (t, occurs + occurs_1, sid_1 \cup sid)$
                1.2.2.  return V
2. V[$pos$+1] $\leftarrow (t, occurs, sid)$  // add to end
3. return V
```

**Figure 41 Datafly VectorAdd algorithm**

```
algorithm suppress(freq, belowk):
// This algorithm suppresses the tuples within freq that do not satisfy the k requirement; these
// should total belowk number of tuples.
// Assume freq has no more than loss * |PT| tuples to suppress, and loss * |PT| = k.
1.  let smallest ← |PT|
2.  for pos ← 1 to |freq| do:
          2.1 (t, count,__) ← freq[pos]
          2.2 if count < k then do:
              2.2.1   freq[pos] ← (null, count,__)
                          where null is the suppressed values for the tuple
              2.2.2. belowk ← belowk – count
          2.3 else do:
              2.3.1   if count < smallest then do:
                      2.3.1.1  smallest ← count
3   if (belowk > 0) and (belowk < k) then do:    // Note. loss * |PT| = k, belowk ≤ k
          3.1 (t, count,__) ← freq[smallest]
          3.2 if (count – belowk) ≥ k then do:
              3.2.1   freq[pos+1] ← (t, count-belowk,__)
              3.2.2   freq[smallest] ← (null, belowk,__)
          3.3 else do:
              3.3.1   freq[smallest] ← (null, count,__)
4   return freq
```

```
algorithm reconstruct(freq):
// This algorithm produces a table based on the tuples within freq and their reported frequencies.
1. let T ← ∅    // T is a table and so it is a multiset, which maintains duplicates
3.  for pos ← 1 to |freq| do:
          4.1 (t, count,sid) ← freq[pos]
          4.2 for each id∈ sid do:
              4.2.1   T ← T ∪ { t[QI, id]}
5   return T
```

**Figure 42 suppress() and reconstruct(), supporting methods for core Datafly algorithm**

Figure 39 lists the core Datafly algorithm. It contains only a few major steps. Step 1 through step 3 construct a frequency list containing unique sequences of values across the quasi-identifier in PT, along with the number of occurrences of each sequence. The frequency list, freq, stores the result. Therefore, each tuple in freq is unique and |freq| ≤ |PT|. The *generalize()* method of sub-step 6.1 is listed in Figure 40. It uses a heuristic to guide its generalization strategy. Specifically, the attribute having the most number of distinct values in the tuples stored in freq is selected. All the values associated with that attribute are generalized, enforcing generalization at the attribute level.

Step 7 assumes that the number of tuples to suppress is less than or equal to *loss* * |PT|. That is, the frequencies associated with tuples in freq that are less than *k*, together total no more than *loss* * |PT|. The *suppress()* method in sub-step 7.1 can be found in Figure 42. It traverses through the tuples of freq replacing the tuples whose frequencies are less than *k* with suppressed values for all the attributes of those tuples, thereby suppressing those tuples. Suppression is enforced at the tuple-level. Complimentary suppression is performed so that the number of suppressed tuples adheres to the *k* requirement. The *reconstruct()* method in sub-step 7.2 can also be found in Figure 42. It produces a table, which becomes MGT, based on freq. Specifically, the values stored for each tuple in freq appear in MGT as they do in freq and are replicated in MGT based on the stored frequency. Therefore, |PT| = |MGT|.

While the core Datafly algorithm is a simplification of the Datafly system that works only across the attributes of the quasi-identifier QI, it can be extended easily to have the generalized table include attributes not in the quasi-identifier. This can be done by assigning a unique identifier to each tuple in PT and then storing along with each tuple in freq, the unique identifiers of the corresponding tuples in PT. The unique identifiers are stored in freq but are not modified or included in step 1 through step 7.1 of the core Datafly algorithm. The *reconstruct()* method in sub-step 7.2 however, is modified to link each tuple from freq to corresponding tuples in PT using the unique identifiers and thereby expand the tuples stored in T to include the additional unchanged attributes of PT that do not belong to QI.

| Race | BirthDate | Gender | ZIP | Problem |
|---|---|---|---|---|
| black | 1965 | male | 02141 | short of breath |
| black | 1965 | male | 02141 | chest pain |
| black | 1965 | female | 02138 | painful eye |
| black | 1965 | female | 02138 | wheezing |
| black | 1964 | female | 02138 | obesity |
| black | 1964 | female | 02138 | chest pain |
| white | 1964 | male | 02139 | obesity |
| white | 1964 | male | 02139 | fever |
| white | 1967 | male | 02138 | vomiting |
| white | 1967 | male | 02138 | back pain |

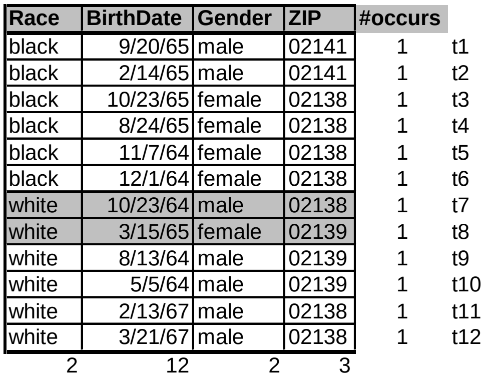
**Figure 43 Table MGT resulting from Datafly, *k*=2, QI={*Race, Birthdate, Gender, ZIP*}**

**Example.**

The private table PT shown in Figure 34 includes unique labels, *t1* through *t12*, associated with each tuple. These labels are useful for linking the Datafly generalization to the original table. Given PT and the domain generalization hierarchies based on the depictions in Figure 33 (on page 101), the core Datafly algorithm provides the table MGT, as shown in Figure 43, as a generalization of PT over the quasi-identifier QI = {*Race*, *BirthDate*, *Gender*, *ZIP*} with no more than *loss*= *k*/|PT|, which is 2/12 (or 17%) of the tuples of PT suppressed. MGT adheres to a *k*-anonymity requirement of *k*=2. Here is a walk through the Datafly algorithm as it constructs MGT.

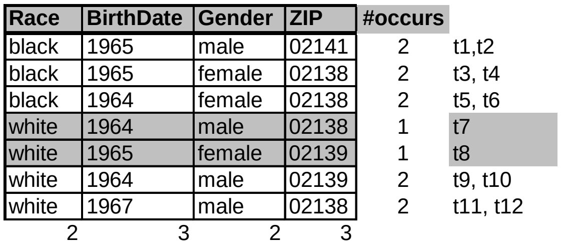
Figure 44 shows the contents of freq after step 3 of the core Datafly algorithm, before any generalization is performed. The sequences of values, considered as a unit across QI in freq, are each unique. The numbers appearing below each column in the tabular view of the attributes of QI in freq report the number of distinct values found in each attribute of QI in freq. For example, there are 2 distinct values, namely "black" and "white" associated with the attribute *Race*; there are 12 distinct values associated with *BirthDate*; 2 with *Gender*; and, 3 with *ZIP*.

In Figure 44, the *BirthDate* attribute has the largest number of distinct values (12) of any attribute of QI in freq; so, at sub-step 6.1, the *generalize()* method re-codes those values to month and year of birth in accordance with the domain generalization hierarchy associated with *BirthDate*. On the second iteration of steps 4 through 6, the *BirthDate* attribute again has the largest number of distinct values (12) of any attribute of QI in freq; so again, these values are recoded. This time values associated with BirthDate report only the year of birth, as shown in Figure 45. The two tuples identified as *t7* and *t8* in Figure 45 do not occur *k* times (only once each). In order for this generalization to be a solution, these two tuples in freq would have to be suppressed. That would be 2/12 (or 17%) of the tuples in PT, which is in accordance with the allowable loss of tuples due to suppression (based on *loss*). Therefore, a solution is found. Figure 43 shows the final result.



| Race | BirthDate | Gender | ZIP | #occurs | |
|---|---|---|---|---|---|
| black | 9/20/65 | male | 02141 | 1 | t1 |
| black | 2/14/65 | male | 02141 | 1 | t2 |
| black | 10/23/65 | female | 02138 | 1 | t3 |
| black | 8/24/65 | female | 02138 | 1 | t4 |
| black | 11/7/64 | female | 02138 | 1 | t5 |
| black | 12/1/64 | female | 02138 | 1 | t6 |
| white | 10/23/64 | male | 02138 | 1 | t7 |
| white | 3/15/65 | female | 02139 | 1 | t8 |
| white | 8/13/64 | male | 02139 | 1 | t9 |
| white | 5/5/64 | male | 02139 | 1 | t10 |
| white | 2/13/67 | male | 02138 | 1 | t11 |
| white | 3/21/67 | male | 02138 | 1 | t12 |
| 2 | 12 | 2 | 3 | | |

**Figure 44 freq at an intermediate stage of the core Datafly algorithm**



| Race | BirthDate | Gender | ZIP | #occurs | |
|---|---|---|---|---|---|
| black | 1965 | male | 02141 | 2 | t1,t2 |
| black | 1965 | female | 02138 | 2 | t3, t4 |
| black | 1964 | female | 02138 | 2 | t5, t6 |
| white | 1964 | male | 02138 | 1 | t7 |
| white | 1965 | female | 02139 | 1 | t8 |
| white | 1964 | male | 02139 | 2 | t9, t10 |
| white | 1967 | male | 02138 | 2 | t11, t12 |
| 2 | 3 | 2 | 3 | | |

**Figure 45 freq at another intermediate stage of the core Datafly algorithm**

## 6.3 Comparison to MinGen

A comparison to MinGen [94] requires examining: (1) the computational complexity of the algorithm to ensure it operates in reasonable time; (2) the correctness of the algorithm in terms of *k*-anonymity protection; and, (3) whether the algorithm distorts minimally. These are discussed in the following subsections.

### 6.3.1 Complexity of the core Datafly algorithm

The core Datafly algorithm listed in Figure 39 with supporting methods in Figure 40 and Figure 42 was not written as efficiently as possible. Nevertheless, here is a walk through the algorithm noting the computational complexity of each part. Its computational complexity is governed by step 4 through step 6 of the core Datafly algorithm. In the worst case, where |freq| = |PT| on the first iteration, step 5 executes |PT| times on the first iteration and fractions of |PT| on subsequent iterations. The construction of a frequency list requires visiting each element of a frequency list and if changes are made due to generalization, the element is removed and then the modified element added. In order to avoid duplication of elements in a frequency list, all elements in the frequency list are compared to the element that is to be inserted. If the elements of freq were stored in a binary tree, then such a comparison could be done in *log*(|freq|) time. In the worst case, |freq| = |PT|; in all cases, |freq| ≤ |PT|. Similarly, in this case, step 6 executes the *generalize()* method in $O(\ |QI| \bullet |PT|\ log\ |PT|)$, if freq was stored as a binary tree, or $O(\ |QI| \bullet |PT|^2)$ as the methods are written. The outer loop from step 4 through step 6 executes $\sum_{i=1}^{|QT|} |DGH_{Ai}|$ times in its worst case, which requires each attribute to generalize one step at a time to its maximal element. So, the overall complexity for the core Datafly algorithm in general is $O\left(\left(\sum_{i=1}^{|QT|} |DGH_{Ai}|\right) \bullet |PT|\right)$. In most databases, |QI| << |PT| and $\sum_{i=1}^{|QT|} |DGH_{Ai}|$ << |PT|. So, the overall complexity for the core Datafly algorithm in general is $O(\ |QI| \bullet |PT|\ log\ |PT|)$, if freq was stored as a binary tree, or $O(\ |QI| \bullet |PT|^2)$ as the methods are written. In comparison to the computational complexity of MinGen [95] and Equation 1 (on page 87), the computational complexity of the core Datafly algorithm is practical but not extremely fast.

### 6.3.2 Correctness of the core Datafly algorithm

The correctness of the core Datafly algorithm relies on its ability to produce solutions that adhere to a given *k*-anonymity requirement, assuming of course a proper quasi-identifier and a proper value for *k* have been provided. In this subsection, I will show that the core Datafly algorithm provides solutions that correctly adhere to a given *k*-anonymity requirement.

The enforcement of a *k*-anonymity requirement is based on step 5, step 6 and step 7 of the core Datafly algorithm. At the conclusion of step 5, the following assertion is true: *belowk* stores the total number of tuples not adhering to the *k*-anonymity requirement. Assume loss has not been inflated. Its minimal required value, based on the stated assumptions by the algorithm, is *loss* * |PT| = *k*. Then, step 6 executes in all cases where *belowk* > *k*, and iteratively generates attributes until *belowk* ≤ *k*. The convergence is assured by the singleton maximal element constraint on each domain generalization hierarchy [96]. Therefore, step 7 executes only if *belowk* ≤ *k*. Sub-step 7.1 executes the *suppress()* method. There are two cases to consider – namely, when *belowk* = *k* and when *belowk* < *k*.

Case 1. If *belowk* = *k*, step 2 of the *suppress()* method will provide suppressed values in freq for what corresponds to *k* tuples in the final table, and these tuples are exactly those tuples for which *belowk* corresponds – i.e., the tuples that do not adhere to *k*-anonymity.

Case 2. If *belowk* < *k*, step 2 of the *suppress()* method behaves as described in case 1 above, except the tuples with suppressed values in freq will themselves not total *k* occurrences. Therefore, the suppressed tuples do not themselves adhere to *k*-anonymity. In this case, additional tuples are suppressed so that the total number of suppressed tuples adhere to the *k*-anonymity requirement. The tuples selected for such complementary suppression come from tuples in freq that already adhere to the *k*-anonymity requirement. In the *suppress()* method listed in Figure 42, a tuple in freq which adheres to the *k*-anonymity requirement and has the fewest number of occurrences in the resulting table is selected. It's position in freq is denoted by *smallest*. [In the full-blown version of the Datafly System, the data holder selects whether a tuple with the fewest, or with the most number of occurrences is used.] In an effort to minimize the suppression, if freq[*smallest*] has at least *k* + (*k-belowk*) occurrences, then only (*k-belowk*) occurrences are suppressed. All tuples in the resulting table therefore, have at least *k* indistinguishable tuples occurring over QI.

### 6.3.3 Summary data attack thwarted by the core Datafly algorithm

The enforcement of the *k*-anonymity requirement even on suppressed tuples protects Datafly from an inference attack based on summary data. If the frequencies of values contained within the privately held information are released separately for each attribute, which is often the case in statistical reports and summary data, then this information can be used to infer suppressed values if the suppressed values themselves do not adhere to the *k*-anonymity requirement imposed on the other released values.

**Example.**

Summary data for the privately held information PT in Figure 34 is shown in Figure 46. Suppose table T in Figure 47 was released as a generalization of PT that satisfied a *k*-anonymity requirement where *k*=2 over the quasi-identifier QI={*Race*, *BirthDate*, *Gender*, *ZIP*}. Except for the single suppressed tuple, *k*-anonymity is satisfied for all the other tuples. However, using the summary data, the missing tuple can be inferred exactly. To combat this problem, the *k*-anonymity requirement must be satisfied on all values, including suppressed ones. The Datafly solution shown in Figure 43 does not have this problem.

| **Race** | **Frequency** |
|---|---|
| black | 6 |
| white | 6 |
| **BirthYear** | |
| 1964 | 5 |
| 1965 | 5 |
| 1967 | 2 |
| **Gender** | |
| male | 6 |
| female | 6 |
| **Problems** | |
| back pain | 1 |
| chest pain | 2 |
| fever | 1 |
| hypertension | 1 |
| obesity | 2 |
| painful eye | 1 |
| short of breath | 2 |
| vomiting | 1 |
| wheezing | 1 |

**Figure 46 Summary data for PT in Figure 34**

| Race | BirthDate | Gender | ZIP | Problem |
|---|---|---|---|---|
| black | 1965 | male | 0214* | short of breath |
| black | 1965 | male | 0214* | chest pain |
| black | 1965 | female | 0213* | painful eye |
| black | 1965 | female | 0213* | wheezing |
| black | 1964 | female | 0213* | obesity |
| black | 1964 | female | 0213* | chest pain |
| white | 1964 | male | 0213* | short of breath |
| white | 1964 | male | 0213* | obesity |
| white | 1964 | male | 0213* | fever |
| white | 1967 | male | 0213* | vomiting |
| white | 1967 | male | 0213* | back pain |
| | | | | |

**Figure 47 Generalization of PT in Figure 34**

### 6.3.4 Distortion and the core Datafly algorithm

In terms of assessing the quality of generalized data that adhere to a *k*-anonymity requirement, it is important to note whether: (1) the resulting data are minimally generalized – i.e., not a generalization of another generalization that satisfies the same *k*-anonymity requirement; and, (2) the data are minimally distorted – i.e., of all minimal generalizations that satisfy the *k*-anonymity requirement, none have more precision retained in the data. In this subsection I will show that the core Datafly algorithm does not necessarily provide minimally generalized solutions or minimally distorted ones, even though its solutions do adhere to a *k*-anonymity requirement.

One of the problems is that Datafly makes crude decisions – generalizing all values associated with an attribute or suppressing all values within a tuple. Algorithms that make decisions at the cell-level can potentially provide better results.

**Example.**

Given the privately held information PT in Figure 34, the Figure 43 provides table MGT, where Datafly(PT)=MGT for *k*=2, quasi-identifier QI={*Race*, *BirthDate*, *Gender*, *ZIP*}, and $\forall i$=1,…,|QI|, $\text{DGH}_{\text{Ai}}$ are domain generalization hierarchies based on the depictions in Figure 33. The precision, *Prec*(MGT) with respect to $\text{DGH}_{\text{Ai}}$ is 0.750. In comparison, Figure 35 provides GT1, where MinGen(PT)=GT1. It is a *k*-minimal distortion of PT over QI with respect to $\text{DGH}_{\text{Ai}}$ where *Prec*(GT1)=0.83. The MinGen result therefore has less distortion based on cell-level generalization and suppression.

Another problem is the heuristic that guides the core Datafly algorithm's selection of which attribute to generalize. The approach of selecting the attribute with the greater number of distinct values, as is done in the *generalize()* method, may be computationally efficient, but can easily lead to unnecessary generalization. Any attribute that is not in the domain of its maximal element could be selected for generalization, though some choices are better than others. The heuristic used in the core Datafly algorithm makes the assumption that having more distinct values associated with an attribute in a table is a perfect predictor of the distance between tuples and of the optimal generalization strategy [97]. Neither of these assumptions is valid. As a result, the core Datafly algorithm can provide more generalization than is needed.

**Example.**

Given the privately held information PT and the generalizations of PT named $\text{GT}_{[1,0]}$, $\text{GT}_{[1,1]}$, $\text{GT}_{[0,2]}$ and $\text{GT}_{[0,1]}$ in **Figure 27**, $\text{GT}_{[1,0]}$, $\text{GT}_{[1,1]}$ and $\text{GT}_{[0,2]}$ all satisfy a *k*-anonymity requirement where *k*=3, the quasi-identifier is QI={*Ethnicity*, *ZIP*}, and where $\forall i$=1,…,|QI|, $\text{DGH}_{\text{Ai}}$ are domain generalization hierarchies based on the depictions in **Figure 26** but where a domain containing the single suppressed value has been affixed atop each. The first iteration of the core Datafly algorithm would provide $\text{GT}_{[0,1]}$ because there are 3 distinct values for *Ethnicity* and 4 distinct values for *ZIP* in PT. However, $\text{GT}_{[0,1]}$ does not satisfy the *k*-anonymity requirement, so another iteration occurs. There are 3 distinct values for *Ethnicity* and 2 distinct values for *ZIP* in $\text{GT}_{[0,1]}$, so $\text{GT}_{[1,1]}$ emerges as the Datafly solution. This table does satisfy the *k*-anonymity requirement. However, $GT_{[1,0]}$ also satisfies the *k*-anonymity requirement and it has less generalization. In fact, $GT_{[1,0]} \leq$ Datafly(PT)=$GT_{[1,1]}$.

## 6.4 Datafly as an anonymous data system

Datafly uses the following disclosure limitation techniques: de-identification, equivalence class substitution, generalization, and suppression. Below is a description of the framework in which Datafly operates.

S = {*subjects whose information is included in PT*}
P = *set of all people whose information could possibly be in PT*
PT = *privately held information about S*
*QI = set of attributes with replications in E*
U = {*existence of people implied by equivalence class assignments*} $\cup P$
RT = *Datafly*(PT)
E = set of publicly available information in today's society
G = set of standard communication methods.
*f = Datafly System*

The system **A**(S, P, PT, *QI*, U, {RT}, E, G, *Datafly*) is an $\mathbf{ADS_{0.}}$

Informal proof.
If *QI* contains all attributes replicated in E, **A** adheres to *k*-map protection, where *k* is enforced on RT. That is, for each value of *QI* released in RT, there are at least *k* tuples having that value, including suppressed tuples; for completeness, see earlier discussion [98].
So, **A** is an $\mathbf{ADS_{0.}}$

Datafly is an $\mathbf{ADS_0}$ in cases where the quasi-identifier is correctly chosen because in those cases each tuple released by Datafly will indistinctly map to at least *k* entities.

## 6.5 Future work

1. The core Datafly algorithm does not typically provide *k*-minimal generalizations [99]. Revise the core Datafly algorithm, or construct a similar algorithm, that makes decisions based on

enforcing generalization at the attribute level and suppression at the tuple level and that operates in real-time, yet provides *k*-minimal generalizations.

2. Similar to the item above, revise the core Datafly algorithm, or construct a similar algorithm, that makes decisions based on enforcing generalization at the attribute level and suppression at the tuple level and that operates in real-time, yet provides *k*-minimal distortions [100] based on a precision metric [101] specific to attribute level generalization and tuple level suppression.

3. The core Datafly algorithm relies on a heuristic to guide its generalization strategy. This heuristic selects the attribute of the quasi identifier having the greater number of distinct values in the modified table as the attribute to generalize. As was discussed earlier [102], this heuristic is computationally efficient but provides no protection against unnecessary generalization. There are many other heuristics that are just as computationally efficient. Perform an analysis that compares a set of such heuristics (including the random selection of an attribute) to optimal results. A nearest neighbor strategy based on distance vectors is used in the *k*-similar algorithm, which appears in a subsequent chapter [103]; perhaps it could be adapted to attribute-level generalization and tuple-level suppression.

4. The core Datafly algorithm presented in Figure 39 was not written to be as computationally efficient as possible. For example, given a private table PT, the original Datafly system could operate in $O(\ |PT|\ \log |PT|\ )$ time. Examine the core Datafly algorithm and its supporting algorithms and improve their computational complexity or prove the minimum complexity required for this approach. Examine and describe best case, worst case and general case scenarios.

# Chapter 7 Results: μ-Argus

In 1996, The European Union began funding an effort that involves statistical offices and universities from the Netherlands, Italy and the United Kingdom. The main objective of this project is to develop specialized software for disclosing public-use data such that the identity of any individual contained in the released data cannot be recognized. Statistics Netherlands has already produced a first version of a program named μ-Argus that seeks to accomplish this goal [104]. The μ-Argus program is considered by many as the official confidentiality software of the European community. A presentation of the concepts on which μ-Argus is based can be found in Willenborg and De Waal [105]. μ-Argus is surprisingly similar to my Datafly system even though the systems were developed at roughly the same time with no prior knowledge of each other and the systems are from different academic traditions. In comparison, as you will see, Datafly tends to over-distort data while μ-Argus tends to under-protect data.

## 7.1 Overview of the μ-Argus System

The program μ-Argus, like the Datafly System, provides protection by enforcing a *k* requirement on the values found in a quasi-identifier. It generalizes values within attributes as needed, and removes extreme outlier information from the released data. The data holder provides a value of *k* and specifies which attributes are sensitive by assigning a value to each attribute between 0 and 3 denoting "not identifying," "most identifying," "more identifying," and "identifying," respectively. The program then identifies rare and therefore unsafe combinations by testing some 2- or 3-combinations of attributes declared to be sensitive. Unsafe combinations are eliminated by generalizing attributes within the combination and by local cell suppression. Rather than removing entire tuples when one or more attributes contain outlier information as is done in the Datafly System, the μ-Argus System simply suppresses or blanks out the outlier values at the cell-level. The resulting data typically contain all the tuples and attributes of the original data, though values may be missing in some cell locations.

Each unique combination of values found within sensitive attributes constitutes a bin. When the number of occurrences of such a combination is less than the minimum required bin size (also known as a *k* requirement), the combination is considered unique and termed an outlier. Clearly for all combinations that include unique identifiers like Social Security numbers, all such combinations are unique. Values associated with outliers must be generalized or one value from each outlier combination must be suppressed. For optimal results when suppression is performed, the program should suppress values that occur in multiple outliers giving precedence to the value occurring most often.

The responsibility of when to generalize and when to suppress lies with the data holder. For this reason, the μ-Argus program operates in an interactive mode so the data holder can see the effect of generalizing and can then select to undo the step. Once a data holder decides to suppress, the selection of which cells require suppression is performed automatically by the program. This is in sharp contrast to Datafly, which automatically produces a complete solution based on data holder specifications. In μ-Argus, a data holder is not even notified whether a current solution satisfies a *k* requirement across the quasi-identifier, so the data holder can easily continue and overly distort data or stop prematurely and under protect data. In addition, there are many possible ways a data holder could rank identifying attributes, and unfortunately different identification ratings typically yield drastically different results. So, ratings and results reported on μ-Argus in this book are based on the most secure possible using the μ-Argus program and therefore, reported use of μ-Argus assumes an extremely knowledgeable data holder.

μ-Argus only uses attribute-level generalization and cell-level suppression. Equivalence class substitution is not provided, as was with Datafly, so the ability to link data across tables to the same person is lost without consistent replacement of identifiers, which provide such links. In fairness to μ-Argus, the current version does not work across multiple tables and as a result it does not take into account many related issues including facilities for longitudinal studies, analysis of the number of records per person, etc, but future versions may do so.

## 7.2 Abstract of the μ-Argus System

I have not found an algorithmic description of μ-Argus in conversation with or in publication by Statistics Netherlands or any other party. Textbook descriptions of how generalization, which they term re-coding, and cell suppression work, as well as instructions and examples of the use of μ-Argus, and a copy of the software were graciously provided by Statistics Netherlands. From these, I have reverse engineered μ-Argus and produced the μ-Argus algorithm shown in Figure 50 with supporting methods found in Figure 51 through Figure 62. By "reverse engineering", I mean that the names of methods and implementation specifics reported in Figure 50 through Figure 62 are created by me in such a way that the overall behavior of each phase of the program agrees, except where noted, with the actual μ-Argus program when provided the same information. The primary phases of the μ-Argus algorithm are provided in Figure 48. During this process of reverse engineering and construction of the algorithm, several shortcomings of the actual μ-Argus implementation were found and are discussed. So in reality, the μ-Argus algorithm I provide in Figure 50 and supporting methods generates solutions that are better protected than those released by the actual program.

Primary phases in the μ-Argus algorithm are as follows:

A. Automatically generalize each attribute independently until it adheres to *k*.

B. Automatically test 2- and 3- combinations of attributes and note outliers.

C. Data holder decides whether to generalize an attribute and if so, identifies the attribute to generalize.

D. Repeat steps B and C until data holder has no more attributes to generalize.

E. Automatically suppress values that occur in multiple outliers, where precedence is given to the value occurring most often.

**Figure 48 Primary phases of μ-Argus algorithm**

The basic phases of the μ-Argus algorithm are listed in Figure 48. The program begins in phase A by automatically generalizing each attribute independently until each value associated with an attribute appears at least *k* times. In phase B, the program then automatically tests combinations of attributes to identify those combinations of attributes whose assigned values in combination do not appear at least *k* times; such combinations of values are termed outliers. Afterwards, the data holder, in phase C, decides whether to generalize an attribute and if so, identifies the attribute to generalize. Phases B and C repeat until the data holder no longer selects an attribute to generalize. Finally, the program in phase E, automatically suppresses values that occur in multiple outliers, where precedence is given to the value occurring most often.

One shortcoming of the actual μ-Argus implementation appears in phase B in Figure 48. Attributes considered sensitive or likely candidates for linking are rated as being either "most identifying" (*Most*), "more identifying" (*More*), or "identifying (*Identifying*) by the data holder. In general, the μ-Argus approach concerns examining 2- and 3- combinations across these classes of attributes. However, μ-Argus does not actually test all 2- and 3- combinations. Figure 49 reports which combinations μ-Argus does and does not test. It is easy to envision situations in which unique combinations appear in combinations not examined by μ-Argus.

| **Combination** | **μ-Argus Tests** |
|---|---|
| *Identifying* × *Identifying* × *Identifying* | **No** |
| *Identifying* × *Identifying* × *More* | **No** |
| *Identifying* × *Identifying* × *Most* | **No** |
| *Identifying* × *More* × *More* | **No** |
| *Identifying* × *More* × *Most* | Yes |
| *Identifying* × *Most* × *Most* | Yes |
| *More* × *More* × *More* | **No** |
| *More* × *More* × *Most* | only if \|*Identifying*\| > 1 |
| *Most* × *Most* × *More* | only if \|*Identifying*\| > 1 |
| *Most* × *Most* × *Most* | Yes |
| *Identifying* × *Identifying* | **No** |
| *Identifying* × *More* | Yes |
| *Identifying* × *Most* | Yes |
| *More* × *More* | only if \|*Identifying*\| > 1 |
| *More* × *Most* | Yes |
| *Most* × *Most* | Yes |

**Figure 49 Combinations of *More, Most, Identifying* tested by μ-Argus**

Figure 49 shows there are 9 combinations involving each of the classes *Most*, *More* and *Identifying*. However, μ-Argus examines only 8 combinations involving *Most*, 6 involving *More* and 4 involving *Identifying*. So, the sensitivity ranking assigned to an attribute by a data holder relates to the number of combinations that are examined and that include the attribute. Even then however, not all possible combinations are examined. If a class has no attributes, then any combination involving it is not computed. In three cases, the size of *Identifying* determines whether combinations of attributes that do not even include *Identifying* are checked. For example, if only *Most* and *More* have attributes and *Identifying* is empty, then only the combinations identified as *More* × *Most* and *Most* × *Most* × *Most* are examined.

**Example.**

Let *Most* = {*SSN*}, *Identifying* = {*Birthdate*, *Gender*, *ZIP*} and *More* be empty. In this case, only *Identifying* × *Most* 2-combinations are examined. Yet, 87% of the population of the United States is considered uniquely identified by {*Birthdate*, *Gender*, *ZIP*}. [106]

Figure 50 contains my description of the μ-Argus algorithm. Figure 51 through Figure 62 provide supporting methods. A description of the general operation of the algorithm and an example using these algorithms are provided following the listings.

**μ-Argus Algorithm**

**Input:** Private Table **PT**; quasi-identifier QI = ($A_1$, …, $A_n$), *k*-anonymity constraint *k*; domain generalization hierarchies $\text{DGH}_{Ai}$, where *i*=1,…,*n* with accompanying functions $f_{Ai}$, and *Most*, *More* and *Identifying*, which are disjoint sets of attributes over the quasi-identifier QI.

**Output:** MT a generalization of PT[QI]

**Assumes:** *Most*, *More* and *Identifying* are disjoint divisions of the attributes over the quasi-identifier QI. That is, QI= $Most \cup More \cup Identifying$ and $Most \cap More = \emptyset$ and $Most \cap Identifying = \emptyset$ and $More \cap Identifying = \emptyset$. PT includes an attribute *id* that serves as a unique identifier (or key) for each tuple in PT.

**algorithm μ-Argus:**

```
// Construct a frequency list containing unique sequences of values across the quasi-identifier in PT,
// along with the number of occurrences of each sequence and the id's of the tuples having that sequence.
    1. let freq be an expandable and collapsible Vector with no elements initially. Each element is of
       the form (QI, frequency, SID, outliers), where $SID = \{id_i : \exists t[id] \in PT[id] \Rightarrow t[id]=id_i\}$; frequency
       = |SID|; and, outliers = $\emptyset$. Therefore, freq is also accessible as a table over (QI, frequency, SID,
       outliers).
    2. freq ← freqSetup(freq, PT, QI)
// generalize each attribute of QI to adhere to k
    3. for each $a_j \in$ QI do:
          3.1. let V be a frequency list of the same type as freq
          3.2. V ← freqConstruct($a_j$)
          3.3. if freqMin(V) < k then do:
                    3.3.1. freq ← generalize($a_j$)
                    3.3.2. go to step 3.2
// check 2- and 3- combinations across Most, More, Identifying
    4. CombinationTest(Most, More, Identifying)
    5. ReportOutliers(freq)   // show data holder outliers
    6. while (data holder wants to generalize an attribute $a_j$) do:
          6.1. freq ← generalize($a_j$)
    7. if (data holder is not done) then do:
          7.1. freq ← ResetOutliers(freq)
          7.2. go to step 4
// suppress outliers and end
    8. freq ← SuppressOutliers(freq)
    9. MT ← reconstruct(freq)
    10. return MT
```

**Figure 50 μ-Argus algorithm**

**μ-Argus freqSetup Algorithm**

**Input:** freq, PT, QI

**Output:** Updates and returns freq, a frequency list

*// This method constructs a frequency list from* PT *based on* QI.

```
algorithm freqSetup:
1.  let $pos \leftarrow 1$, $total \leftarrow 0$
2.  while total $\neq$ |PT| do
    2.1.  freq[$pos$] $\leftarrow$ ($t$[QI], $occurs$, $sid$, $\emptyset$)
              where $t$[QI]$\in$PT[QI], ($t$[QI],__)$\notin$ freq,
                  $occurs$ = |PT| - |PT[QI] – {$t$[QI]}|
                  $sid$ = {$t$[$id$] : $t$[QI,$id$]$\in$PT[QI,$id$]
    2.2.  $pos \leftarrow pos + 1$, $total \leftarrow total + occurs$
3.  return freq
```

**Figure 51 μ-Argus freqSetup algorithm**

**μ-Argus freqConstruct Algorithm**

**Input:** $A_x$, …, $A_y$, which is a list of one or more attributes and each such attribute is an element of QI.

**Output:** V, a frequency list based on freq[$A_x$,…,$A_y$].

**Assumes:** $A_x$,…,$A_y$ contains no duplicates and each is a member of QI and freq is available for use.

*// This algorithm constructs a frequency list from the tuples of* freq *over a subset of attributes of* QI.

```
algorithm freqConstruct:
    1.  let V be a frequency list of the same type as freq. V initially has no elements.
    2.  for $pos \leftarrow$ 1 to |freq| do:
        1.1. ($t$, $occurs$, $sid$, $outliers$) $\leftarrow$ freq[$pos$]
        1.2. V $\leftarrow$VectorAdd(V, $t$[$A_x$,…,$A_y$], $occurs$, $sid$)
    3.  return V
```

**Figure 52 μ-Argus freqConstruct algorithm**

**μ-Argus VectorAdd Algorithm**

**Input:** V, *t*, *occurs*, *sid*

**Output:** Updates and returns V, a frequency list

*// This method adds the tuples associated with (*t,occurs,sid*) to* V *avoiding duplication*

**algorithm VectorAdd:**

```
2.  for $pos \leftarrow 1$ to $|V|$ do:
        3.4. let $(t_1, occurs_1, sid_1, outliers_1) \leftarrow V[pos]$
        3.5. if $t_1 = t$ then do:
            3.5.1.  $V[pos] \leftarrow (t, occurs + occurs_1, sid_1 \cup sid, \emptyset)$
            3.5.2.  return V
4.  $V[pos+1] \leftarrow (t, occurs, sid, \emptyset)$  // add to end
5.  return V
```

**Figure 53 μ-Argus VectorAdd algorithm**

**μ-Argus freqMin Algorithm**

**Input:** V, a frequency list based on freq[$A_x$,…,$A_y$].

**Output:** an integer reporting the smallest number of *occurs* in V

*// This method returns the minimum number of occurrences in* V

**algorithm freqMin:**

```
    1.  let $min \leftarrow |PT|$
    2.  for $pos \leftarrow 1$ to $|V|$ do:
        1.1.  $(t, occurs, sid, outliers) \leftarrow V[pos]$
        1.2.  if $occurs < min$ then do:
            1.2.1.  $min \leftarrow occurs$
    3.  return $min$
```

**Figure 54 μ-Argus freqMin algorithm**

```
μ-Argus generalize Algorithm
Input:      attr, which is an attribute of QI
Output:     updates and returns freq
Assumes:    freq and domain generalization hierarchy $DGH_{attr}$ with accompanying function $f_{attr}$ are
            available for use; and $attr \in QI$
// This method generalizes all values associated with attr in freq.
algorithm generalize:
    1. let V be a frequency list of the same type as freq. V initially has no elements.
    2. for pos ← 1 to |freq| do:
        2.1. ([$v_{a1}$,…,$v_{an}$], occurs, sid, outliers) ← freq[pos]
        2.2. if $v_{attr}$ is not maximal element of $DGH_{attr}$ then do:
            2.2.1. if $attr = a_1$ then do:
                2.2.1.1. V ← VectorAdd(V, [$f_{attr}(v_{a1})$,…,$v_{an}$], occurs, sid, ∅)
            2.2.2. else if $attr = a_n$ then do:
                2.2.2.1. V ← VectorAdd(V, [$v_{a1}$,…,$f_{attr}(v_{an})$], occurs, sid, ∅)
            2.2.3. else do:
                2.2.3.1. V ← VectorAdd(V, [$v_{a1}$,…,$f_{attr}(v_{attr})$, …,$v_{an}$], occurs, sid, ∅)
    3. freq ← V
    4. return freq
```

**Figure 55 μ-Argus generalize algorithm**

**μ-Argus CombinationTest Algorithm**

**Input:** *Most*, *More* and *Identifying*, which are disjoint sets of attributes over the quasi-identifier QI.

**Output:** Updates and returns outliers in freq.

**Assumes:** *Most*, *More* and *Identifying* are disjoint sets of attributes over the quasi-identifier QI. Each cell of outliers is initialized to ∅ and outliers is available for use.

```
// This method computes 2- and 3- way combinations across Most, More, and Identifying.
// This method selects those combinations the actual μ-Argus program would compute.
// Notice it is not all 2- and 3- combinations.
algorithm CombinationTest:
1.  if $|Most| = 0$ then return ∅
2.  if $|More| = 0$ and $|Identifying| = 0$ then return ∅
// guarantee: $|\text{Most}| \geq 1$
3.  if $|Most| \geq 3$ then do:
        3.1 MarkOutliers3(Most)    // $Most \times Most \times Most$
4.  if $|Most| \geq 2$ then do:
        4.1 MarkOutliers2 (Most, ∅)    // $Most \times Most$
        4.2 if $|Identifying| \geq 1$ and $|More| \geq 1$ then do:
            4.2.1    MarkOutliers2 (Most, More)    // $Most \times Most \times More$
        4.3 if $|Identifying| \geq 1$ then do:
            4.3.1    MarkOutliers2 (Most, Identifying)    // $Most \times Most \times Identifying$
5.  if $|More| \geq 2$ then do:
        5.1 MarkOutliers (Most, More, ∅)    // $Most \times More$
        5.2 if $|Identifying| \geq 1$ then do:
            5.2.1    MarkOutliers2 (More, ∅)    // $More \times More$
            5.2.2    MarkOutliers2 (More, Most)    // $More \times More \times Most$
6.  if $|More| \geq 1$ and $|Identifying| \geq 1$ then do:
        6.1 MarkOutliers (Most, More, Identifying)    // $Most \times More \times Identifying$
        6.2 MarkOutliers (More, Identifying, ∅)    // $More \times Identifying$
7.  if $|Identifying| \geq 1$ then do:
        7.1 MarkOutliers (Most, Identifying, ∅)    // $Most \times Identifying$
8.  return
```

**Figure 56 μ-Argus CombinationTest algorithm**

**μ-Argus MarkOutliers Algorithm**

**Input:** *S1*, *S2*, *S3*, which are subsets of QI

**Output:** Updates *outliers* in freq and returns updated freq

**Assumes:** *S1*, *S2* and *S3* are disjoint sets of attributes over the quasi-identifier QI and freq is available for use.

```
// This method computes the sub-tables S1 x S2 x S3 and marks outliers
algorithm MarkOutliers:
    1.  for $i \leftarrow 1$ to |QI| do:
        1.1. for $j \leftarrow 1$ to |QI| do:
            1.1.1.  if $a_i \in S1$ and $a_j \in S2$ then do:
                1.1.1.1.  if $|S3| = 0$ then do:
                    1.1.1.1.1.  V ← freqConstruct($a_i$, $a_j$)
                    1.1.1.1.2.  freq ← MarginalUpdate(V)
                1.1.2.  else do:
                    1.1.2.1.  for $k \leftarrow 1$ to |QI| do:
                        1.1.2.1.1.1.  if $a_k \in S3$ then do:
                            1.1.2.1.1.1.1.  V ← freqConstruct($a_i$, $a_j$, $a_k$)
                            1.1.2.1.1.1.2.  freq ← MarginalUpdate(V)
    2.  return freq
```

**Figure 57 μ-Argus MarkOutliers algorithm**

**μ-Argus MarkOutliers2 Algorithm**

**Input:** *S1*, *S3*, which are subsets of QI

**Output:** Updates *outliers* in freq and returns updated freq

**Assumes:** *S1*, *S3* are disjoint sets of attributes over the quasi-identifier QI and freq is available for use.

```
// This method computes the sub-tables S1 x S1 x S3 and marks outliers
algorithm MarkOutliers2:
    1.  for $i \leftarrow 1$ to |QI| do:
        1.1. for $j \leftarrow i+1$ to |QI| do:
            1.2.2.  if $a_i \in S1$ and $a_j \in S1$ then do:
                1.2.2.1. if $|S3| = 0$ then do:
                    1.2.2.1.1.  V ← freqConstruct($a_i$, $a_j$)
                    1.2.2.1.2.  freq ← MarginalUpdate(V)
                1.2.2.2. else do:
                    1.2.2.2.1.  for $k \leftarrow 1$ to |QI| do:
                        1.2.2.2.1.1. if $a_k \in S3$ then do:
                            1.2.2.2.1.1.1.  V ← freqConstruct($a_i$, $a_j$, $a_k$)
                            1.2.2.2.1.1.2.  freq ← MarginalUpdate(V)
    2. return freq
```

**Figure 58 μ-Argus MarkOutliers2 algorithm**

```
μ-Argus MarkOutliers3 Algorithm
Input:      S1, which is a subset of QI.
Output:     Updates outliers in freq and returns updated freq
Assumes:    S1 is a non-empty subset of QI and freq is available for use.
// This method computes the sub-tables S1 x S1 x S1 and marks outliers
algorithm MarkOutliers3:
    1. for $i \leftarrow 1$ to |QI| do:
        a. for $j \leftarrow i+1$ to |QI| do:
            i. for $k \leftarrow j+1$ to |QI| do:
                1. if $a_i \in S1$ and $a_j \in S1$ and $a_k \in S1$ then do:
                    a. V ← freqConstruct($a_i$, $a_j$, $a_k$)
                    b. freq ← MarginalUpdate(V)
    4. return freq
```

**Figure 59 μ-Argus MarkOutliers3 algorithm**

```
μ-Argus MarginalUpdate Algorithm
Input:      V, which is a frequency list based on freq[$A_x$,…,$A_y$], and A, which is a set of attributes
            where each attribute is a member of QI.
Output:     Updates outliers in freq and returns the updated freq.
Assumes     A is a non-empty subset of QI and freq is available for use.
// This method records outliers by storing the combination of attributes (A) known not to adhere to k
// in freq.
algorithm MarginalUpdate:
1. for pos ← 1 to |V| do:
    1.1. (t, occurs, sid, outliers) ← V[pos]
    1.2. if occurs < k then do:
        1.2.1. for $pos_1$ ← 1 to |freq| do:
            1.2.1.1. let ($t_1$, $occurs_1$, $sid_1$, $outliers_1$) ← freq[$pos_1$]
            1.2.1.2. if $|sid_1 \cap sid| \geq 1$ then do:
                1.2.1.2.1. $outliers_1 \leftarrow outliers_1 \cup \{A\}$
                1.2.1.2.2. freq[$pos_1$] ← ($t_1$, $occurs_1$, $sid_1$, $outliers_1$)
2. return freq
```

**Figure 60 μ-Argus MarginalUpdate algorithm**

```
μ-Argus resetOutliers Algorithm
Input:      freq
Output:     updates and returns freq
// This method sets all outliers in freq to the empty set.
algorithm resetOutliers:
    1.  for $pos \leftarrow 1$ to |V| do:
            1.1. $(t, occurs, sid, outliers) \leftarrow$ V[$pos$]
            1.2. V[$pos$] $\leftarrow (t, occurs, sid, \emptyset)$
    2.  return freq
```

**Figure 61 μ-Argus resetOutliers algorithm**

```
μ-Argus SuppressOutliers Algorithm
Input:      freq
Output:     Updates and returns freq.
// This method suppresses one value of each combination known to be an outlier in a tuple.
algorithm SuppressOutliers:
1.    for $pos \leftarrow 1$ to |freq| do:
        1.1.  $([v_{a1},\ldots,v_{an}], occurs, sid, outliers) \leftarrow$ freq[$pos$]
        1.2.  if $occurs < k$ then do:
                1.2.1.  while $|outliers| > 0$ do:  // actual μ-Argus program does not exhaust outliers!
                    1.2.1.1.    let $max \leftarrow 0$
                    1.2.1.2.    for $i \leftarrow 1$ to |QI| do:
                        1.2.1.2.1.    let $total \leftarrow 0$
                        1.2.1.2.2.    for each $s \in outliers$ do:
                                1.2.1.2.2.1.  if $a_i \in s$ then do:
                                    1.2.1.2.2.1.1.    $total \leftarrow total + 1$
                        1.2.1.2.3.    if $total > max$ then do:
                                1.2.1.2.3.1.   $max \leftarrow total$
                                1.2.1.2.3.2.   $attr \leftarrow a_i$
                    // attr is most frequent attribute in outliers
                    1.2.1.3.    $outliers \leftarrow \{ s_k : s_k \in outliers, attr \notin s_k \}$
                    1.2.1.4.    if $attr = a_1$ then do:
                        1.2.1.4.1.    freq[$pos$] $\leftarrow$ ([null,$\ldots, v_{an}$], $occurs, sid, outliers$)
                    1.2.1.5.    else if $attr = a_n$ then do:
                        1.2.1.5.1.    freq[$pos$] $\leftarrow$ ([$v_{a1},\ldots,$null], $occurs, sid,$ outliers)
                    1.2.1.6.    else do:
                        1.2.1.6.1.    let $a_j$ be $attr$, where $QI=a_1,\ldots,a_{j-1}, a_j, a_{j+1}\ldots,a_n$
                        1.2.1.6.2.    freq[$pos$] $\leftarrow$ ([$v_{a1},\ldots,a_{j-1}$,null,$a_{j+1}\ldots,v_{an}$], $occurs, sid,$ outliers)
    2. freq $\leftarrow$ freqCleanup(freq)  // consolidates elements to avoid supplicate values over QI
    3.  return freq
```

**Figure 62 μ-Argus SuppressOutliers algorithm**

As introduced earlier, the basic steps, A through E, of the μ-Argus algorithm are enumerated in Figure 48. The algorithm listed in Figure 50 along with its supporting methods is more detailed but follows these same basic steps. Below is a walk through the detailed version of the μ-Argus algorithm.

Given a private table PT, a quasi-identifier QI=($A_1,\ldots,A_n$), a $k$-anonymity requirement $k$, domain generalization hierarchies $\text{DGH}_{\text{Ai}}$, where $i$=1,…,$n$ with accompanying functions $f_{Ai}$, and *Most*, *More* and *Identifying*, which are disjoint sets of attributes over the quasi-identifier QI, the μ-Argus algorithm, listed in Figure 50, generates a generalization of PT[QI]. The algorithm assumes *Most*, *More* and *Identifying* are disjoint divisions of the attributes over the quasi-identifier QI. That is, (QI= *Most* ∪ *More* ∪ *Identifying*) and (*Most* ∩ *More* = ∅) and (*Most* ∩ *Identifying* = ∅) and (*More* ∩ *Identifying* = ∅). PT is also required to have a unique identifier associated with each of its tuples; in this case, PT includes an attribute *id* that serves as a unique identifier (or key) for each tuple in PT.

The μ-Argus algorithm begins in steps 1 and 2 by constructing a frequency list named freq. Conceptually I define a frequency list as simply a vector. But as the primary data structure in this algorithm, my notion of a frequency list is that it describes a table T. Each element in the frequency list freq corresponds to one or more tuples in table T. The frequency list freq begins by describing the table PT and each table T subsequently described in freq is a generalization of PT. Frequency lists are also used to store variations and subsets of freq during the operation of the algorithm.

Each element in a frequency list F based on a table T consists of (1) values assigned to the quasi-identifier [$v_{Ai},\ldots,v_{An}$]; (2) the number of tuples in T, referred to as *occurs*, having that assignment of values; (3) the set of tuple identifiers, referred to as *SID*, in T to which the values $v_{Ai},\ldots,v_{An}$ refer, and, (4) a set of attributes, referred to as *outliers*, that are initially set to the empty set but at one point in the algorithm contain the attributes for which the assigned values occur less than the $k$ requirement warrants. The invariant |*SID*| = *occurs* holds in F. Each sequence of values [$v_{Ai},\ldots,v_{An}$] is unique in F.

Step 1 and step 2 of the μ-Argus algorithm listed in Figure 50 produces a frequency list freq based on the tuples of the privately held table PT. The method *freqSetup()* defined in Figure 51 performs the construction.

Step 3 of the μ-Argus algorithm listed in Figure 50 generalizes each attribute in freq so there are at least *k* occurrences of each value reported for an attribute. Recall an earlier lemma in which a table T that satisfies a quasi-identifier QI=$A_1,\ldots,A_n$ must have at least *k* occurrences of each $t[A_i]\in$ T where $i$=1,…,*n* [107]. Step 3 of the μ-Argus algorithm automatically generalizes attributes until this condition is satisfied. Success is guaranteed by the single maximal element requirement of each domain generalization hierarchy $\text{DGH}_{\text{Ai}}$ where $i$=1,…,*n* [108].

The heart of the μ-Argus algorithm resides in steps 4 through 8 of the listing in Figure 50. These steps concern examining values associated with 2- and 3- combinations of attributes across the quasi-identifier QI. The data holder provides the attributes of QI by providing the 3 sets named *Most*, *More* and *Identifying*. The set named *Most* consists of attributes of QI the data holder considers "most identifying". The set named *More* consists of attributes of QI the data holder considers "more identifying". And, the set named *Identifying* consists of attributes of QI the data holder considers merely "identifying".

In step 4, values associated with 2- and 3- combinations of attributes across *More*, *Most* and *Identifying* are examined to determine which combinations of values do not adhere to the *k* requirement. These values are considered outliers, are the attributes associated with these values are recorded as *outliers* for these tuples in freq. As step 5, these *outliers* are displayed for the data holder to inspect. In the next paragraphs, I described the generation and inspection of these combinations in detail.

The method *CombinationTest()*, listed in Figure 56, generates the 2- and 3- combinations that are examined in μ-Argus. As discussed earlier and listed in Figure 48, the actual μ-Argus program does not examine all 2- and 3- combinations of values across the attributes of *Most*, *More* and *Identifying*. Instead, it examines a subset of these combinations based on the rank order of *More*, *More* and then *Identifying*. The method *CombinationTest()* explores only those combinations examined by the actual μ-Argus program as listed in Figure 49.

The actual work of generating the sub-tables that represent the 2- and 3-combinations and marking the outliers found is done by three methods. These are *MarkOutliers()*, *MarkOutliers2()* and *MarkOutliers3()*. Each of these methods receives a combination based on *Most*, *More*, *Identifying* and ∅ as arguments.

The method *MarkOutliers()*, listed in Figure 57, takes 3 arguments *S1*, *S2*, and *S3*, and computes sub-tables based on $S1 \times S2 \times S3$. Each element in *S1*, *S2* and *S3* is assumed to be an element of QI. The method then explores $S1 \times S2$ if $S3 = \emptyset$ or $S1 \times S2 \times S3$ if $S3 \neq \emptyset$. The arguments *S1* and *S2* are required and cannot be ∅, but *S3* can be ∅. It is assumed that ($S1 \cap S2 = \emptyset$). If S3 ≠ ∅, then it is also assumed that ($S1 \cap S3 = \emptyset$) and ($S2 \cap S3 = \emptyset$).

To make sure duplicate combinations are not explored when examining combinations across the same set, *MarkOutliers2()* and *MarkOutliers3()* are used. The method *MarkOutliers3()*, listed in Figure 59, is used when a 3-combination is explored across a single set of attributes. For example, *Most* × *Most* × *Most* is examined by executing *MarkOutliers(Most)*.

Similarly, *MarkOutliers2()*, listed in Figure 58, is used when a 2- or 3-combination involves repeating the first set. For example, *Most* × *Most* is examined by executing *MarkOutliers2(Most, ∅)* and *Most* × *Most* × *More* is examined by executing *MarkOutliers2(Most, More)*.

The methods *MarkOutliers()*, *MarkOutliers2()* and *MarkOutliers3()* work as follows. First, they generate a frequency list V that contains a sub-table from freq based on values associated with 2 or 3 combinations of the attributes provided as parameters. This is done using the method *freqConstruct()*, which is listed in Figure 52. The method *freqConstruct()* is given a sequence of attributes $A_x,\ldots,A_y$ and generates V from freq[$A_x,\ldots,A_y$]. The methods *MarkOutliers()*, *MarkOutliers2()* and *MarkOutliers3()* then record in freq those combinations of values in V that do not adhere to *k*. This is done using the method *MarginalUpdate()*, which is listed in Figure 60. The method *MarginalUpdate()* records combinations of values associated with $A_x,\ldots,A_y$ in V that do not adhere to the *k* requirement by appending $\{A_x,\ldots,A_y\}$ to the *outliers* of freq for each associated tuple.

In step 5 of the µ-Argus algorithm, which is listed in Figure 50, the tuples and combinations of attributes containing outliers is displayed for the data holder's inspection. The *ReportOutliers()* method, a listing of which is not provided, merely visits each element of freq. If the element has a non-empty value for *outliers*, then the corresponding combinations of attributes contained in *outliers* are displayed. The purpose is for the data holder to decide whether to generalize any attributes or whether to stop execution. The *generalize()* method, which is listed in Figure 55, replaces the values associated with an attribute in freq with their generalized replacement. Step 6 of the μ-Argus algorithm allows the data holder to generalize as many attributes of QI as desired.

At step 7 of the μ-Argus algorithm, which is listed in Figure 50, the data holder can decide to have the 2- and 3- combinations re-analyzed, presumably after some attributes have been generalized. If the combinations are to be re-analyzed, the values associated with *outliers* recorded in freq are initialized to ∅ and execution continues at step 4, thereby repeating steps 4 through 7. The *resetOutliers()* method, which is listed in Figure 61, sets the values associated with *outliers* in freq to the empty set.

Alternatively, the data holder can decide to conclude the program; in which case, step 8 and step 9 of the μ-Argus algorithm, which are listed in Figure 50, execute. Step 8 involves suppressing a value of each combination of values known to be an outlier in a tuple. This is done by executing *SuppressOutliers()*, which is listed in Figure 62. The operation of this method is described below.

The *SuppressOutliers()* method visits each element in freq that does not adhere to the *k* requirement. Clearly, from the operation of the μ-Argus algorithm, it can be shown that each such element will not necessarily have a non-empty *outliers* value because there may exist 4-combinations across QI and there may exist larger combinations of values across QI in the data that are unique. In addition there may exist 2- or 3-combinations across QI that are unique and not identified because those combinations were not examined by *CombinationTest()* at all. These possibilities pose serious problems for the way in which μ-Argus has been implemented.

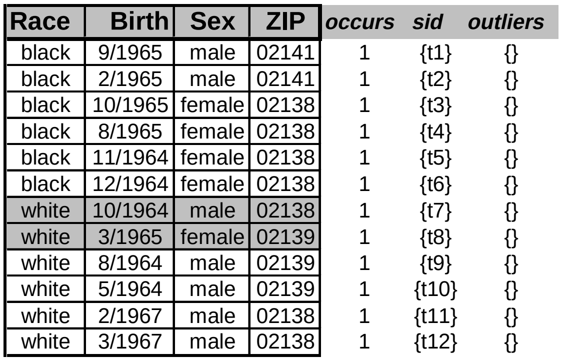
Each element in freq is visited in *SuppressOutliers()*. If the value for *outliers* associated with that element is not empty, then the value associated with an attribute occurring most frequently in that element's *outliers* is suppressed (i.e., a suppressed value is one that is assigned a `null` value in the method). The *while()* loop in the *SuppressOutliers()* method continues in step 1.2.1 until all combinations identified in *outliers* has a value in the combination of values suppressed. This is in sharp contrast to the actual μ-Argus program. In the actual μ-Argus program, each such combination is not exhausted. As a result, some combinations of values whose attributes are identified in *outliers* may not have values suppressed even though all combinations reported in *outliers* is known to not adhere to the *k* requirement. This is obviously a problem with the μ-Argus implementation and not a limitation of its approach.

The final step of the μ-Argus algorithm is to construct a table based on the descriptions of tuples in freq. The *reconstruct()* method, which is listed in Figure 42, works the same in μ-Argus as in Datafly. It can be shown that the final table resulting from the μ-Argus algorithm is a generalization of the original table provided because the only operations on the data were generalization and suppression.

**Example**

The private table PT shown in Figure 34 includes unique labels, *t1* through *t12*, associated with the *id* attribute. These labels are useful for linking the resulting generalization to the original table. Given PT and the domain generalization hierarchies based on the depictions in Figure 33 (on page 101), the μ-Argus algorithm, which is listed in Figure 50, provides the table MT, as shown in Figure 75, as a generalization of PT over the quasi-identifier QI = {*Race*, *BirthDate*, *Gender*, *ZIP*}, where *Most* = {*BirthDate*}, *More* = {*Gender*, *ZIP*} and *Identifying* = {*Race*}. The actual μ-Argus program provides the table MTactual shown in Figure 76 as a generalization of PT over QI. Both MT and MTactual are supposed to adhere to a *k*-anonymity requirement of *k*=2. Here is a walk through the μ-Argus algorithm to demonstrate how MT and MTactual are constructed.

Figure 63 shows the contents of the frequency list freq after step 2 of the μ-Argus algorithm completes. Each sequence of values across QI is unique in PT and so each tuple has a distinct corresponding element in freq.



| Race | Birth | Sex | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|---|
| black | 9/1965 | male | 02141 | 1 | {t1} | {} |
| black | 2/1965 | male | 02141 | 1 | {t2} | {} |
| black | 10/1965 | female | 02138 | 1 | {t3} | {} |
| black | 8/1965 | female | 02138 | 1 | {t4} | {} |
| black | 11/1964 | female | 02138 | 1 | {t5} | {} |
| black | 12/1964 | female | 02138 | 1 | {t6} | {} |
| white | 10/1964 | male | 02138 | 1 | {t7} | {} |
| white | 3/1965 | female | 02139 | 1 | {t8} | {} |
| white | 8/1964 | male | 02139 | 1 | {t9} | {} |
| white | 5/1964 | male | 02139 | 1 | {t10} | {} |
| white | 2/1967 | male | 02138 | 1 | {t11} | {} |
| white | 3/1967 | male | 02138 | 1 | {t12} | {} |

**Figure 63 freq after *freqSetup()* in μ-Argus algorithm step 2**

Figure 64 shows the contents of freq after step 3 of the μ-Argus algorithm completes. Each value associated with each attribute in QI adheres to the *k*–requirement. That is, each value has at least *k* occurrences; in this example, *k*=2. In order to achieve this in freq, values associated with *BirthDate* were generalized to the year of birth.

| Race | Birth | Sex | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|---|
| black | 1965 | male | 02141 | 2 | {t1,t2} | {} |
| black | 1965 | female | 02138 | 2 | {t3,t4} | {} |
| black | 1964 | female | 02138 | 2 | {t5,t6} | {} |
| white | 1964 | male | 02138 | 1 | {t7} | {} |
| white | 1965 | female | 02139 | 1 | {t8} | {} |
| white | 1964 | male | 02139 | 2 | {t9,t10} | {} |
| white | 1967 | male | 02138 | 2 | {t11,t12} | {} |

**Figure 64 freq after generalize loops in μ-Argus algorithm, step 3**

Step 4 of the μ-Argus algorithm executes *CombinationTest()*, which is listed in Figure 56. This method computes 2- and 3-combinations across *Most*, *More* and *Identifying* to determine which combinations of values, if any, do not occur at least *k* times; recall, in this example *k*=2. It begins by examining *Most* × *More* combinations. Figure 65 shows the frequency list V generated by *MarkOutliers()* at step 5.1 in *CombinationTest()* when it examines *BirthDate* × *Sex*. As Figure 65 shows, all combinations of values for these attributes found in freq occur at least *k* times.

| Birth | Sex | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|
| 1965 | male | 2 | {t1,t2} | {} |
| 1965 | female | 3 | {t3,t4,t8} | {} |
| 1964 | female | 2 | {t5,t6} | {} |
| 1964 | male | 3 | {t7,t9,t10} | {} |
| 1967 | male | 2 | {t11,t12} | {} |

**Figure 65 V at *Most* × *More* in *CombinationTest()*, step 5.1**

Continuing the examination of *Most* × *More* combinations, Figure 66 shows the frequency list V generated by *MarkOutliers()* at step 5.1 in *CombinationTest()* when it examines *BirthDate* × *ZIP*. The combination where *BirthDate*="1965" and *ZIP*="02139" occurs only once and appears in the tuple identified as *t8* in PT. As a result, *outliers* in freq is updated to include {*Birthdate, ZIP*} for that element. Depictions of the resulting V and freq tables are shown in Figure 66.

| Birth | ZIP | occurs | sid | outliers |
|---|---|---|---|---|
| 1965 | 02141 | 2 | {t1,t2} | {} |
| 1965 | 02138 | 2 | {t3,t4} | {} |
| 1964 | 02138 | 3 | {t5,t6,t7} | {} |
| 1965 | 02139 | 1 | {t8} | {} |
| 1964 | 02139 | 2 | {t9,t10} | {} |
| 1967 | 02138 | 2 | {t11,t12} | {} |

V

| Race | Birth | Sex | ZIP | occurs | sid | outliers |
|---|---|---|---|---|---|---|
| black | 1965 | male | 02141 | 2 | {t1,t2} | {} |
| black | 1965 | female | 02138 | 2 | {t3,t4} | {} |
| black | 1964 | female | 02138 | 2 | {t5,t6} | {} |
| white | 1964 | male | 02138 | 1 | {t7} | {} |
| white | 1965 | female | 02139 | 1 | {t8} | **{{birth,zip}}** |
| white | 1964 | male | 02139 | 2 | {t9,t10} | {} |
| white | 1967 | male | 02138 | 2 | {t11,t12} | {} |

freq

**Figure 66 freq and V at *Most* × *More* in *CombinationTest()*, step 5.1**

The next combinations examined result from *More* × *More*, but there is only one such combination, namely *Sex* × *ZIP*. Figure 67 shows the frequency list V generated by *MarkOutliers2()* at step 5.2.1 in *CombinationTest()* when it examines *Sex* × *ZIP*. The combination where *Sex*="female" and *ZIP*="02139" occurs only once and appears in the tuple identified as *t8* in PT. As a result, *outliers* in freq is updated to include {*Sex*, *ZIP*} for that element. Depictions of the resulting V and freq tables are shown in Figure 67.

| Sex | ZIP | occurs | sid | outliers |
|---|---|---|---|---|
| male | 02141 | 2 | {t1,t2} | {} |
| female | 02138 | 4 | {t3,t4,t5,t6} | {} |
| male | 02138 | 3 | {t7,t11,t12} | {} |
| female | 02139 | 1 | {t8} | {} |
| male | 02139 | 2 | {t9,t10} | {} |

V

| Race | Birth | Sex | ZIP | occurs | sid | outliers |
|---|---|---|---|---|---|---|
| black | 1965 | male | 02141 | 2 | {t1,t2} | {} |
| black | 1965 | female | 02138 | 2 | {t3,t4} | {} |
| black | 1964 | female | 02138 | 2 | {t5,t6} | {} |
| white | 1964 | male | 02138 | 1 | {t7} | {} |
| white | 1965 | female | 02139 | 1 | {t8} | **{{birth,zip}, {sex,zip}}** |
| white | 1964 | male | 02139 | 2 | {t9,t10} | {} |
| white | 1967 | male | 02138 | 2 | {t11,t12} | {} |

freq

**Figure 67 freq and V at *More* x *More* in *CombinationTest()*, step 5.2.1**

The next combinations examined result from *More* × *More* × *Most*, but there is only one such combination, namely *BirthDate* × *Sex* × *ZIP*. Figure 68 shows the frequency list V generated by *MarkOutliers2()* at step 5.2.2 in *CombinationTest()* when it examines *BirthDate* × *Sex* × *ZIP*. The combination where *BirthDate*="1964", *Sex*="male" and *ZIP*="02138" occurs only once and appears in the tuple identified as *t7* in PT. Likewise, the combination where *BirthDate*="1965", *Sex*="female" and *ZIP*="02139" occurs only once and appears in the tuple identified as *t8* in PT. As a result, *outliers* in freq is updated to include {*BirthDate*, *Sex*, *ZIP*} for those elements. Depictions of the resulting V and freq tables are shown in Figure 68.

| Birth | Sex | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|
| 1965 | male | 02141 | 2 | {t1,t2} | {} |
| 1965 | female | 02138 | 2 | {t3,t4} | {} |
| 1964 | female | 02138 | 2 | {t5,t6} | {} |
| 1964 | male | 02138 | 1 | {t7} | {} |
| 1965 | female | 02139 | 1 | {t8} | {} |
| 1964 | male | 02139 | 2 | {t9,t10} | {} |
| 1967 | male | 02138 | 2 | {t11,t12} | {} |

**V**

| Race | Birth | Sex | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|---|
| black | 1965 | male | 02141 | 2 | {t1,t2} | {} |
| black | 1965 | female | 02138 | 2 | {t3,t4} | {} |
| black | 1964 | female | 02138 | 2 | {t5,t6} | {} |
| white | 1964 | male | 02138 | 1 | {t7} | **{{birth,sex,zip}}** |
| white | 1965 | female | 02139 | 1 | {t8} | **{{birth,zip}, {sex,zip}, {birth,sex,zip}}** |
| white | 1964 | male | 02139 | 2 | {t9,t10} | {} |
| white | 1967 | male | 02138 | 2 | {t11,t12} | {} |

**freq**

**Figure 68 freq and V at *More x More x Most* in *CombinationTest()*, step 5.2.2**

The next combinations examined result from *Most* × *More* × *Identifying*. These are the specific combinations *Race* × *BirthDate* × *Sex* and *Race* × *BirthDate* × *ZIP*. Figure 69 shows the frequency list V generated by *MarkOutliers()* at step 6.1 in *CombinationTest()* when it examines *Race* × *BirthDate* × *Sex*. The combination where *Race*="white", *BirthDate*="1965" and *Sex*="female" occurs only once and appears in the tuple identified as *t8* in PT. As a result, *outliers* in freq is updated to include {*Race*, *BirthDate*, *Sex*} for this element. Depictions of the resulting V and freq tables are shown in Figure 69.

| Race | Birth | Sex | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|
| black | 1965 | m | 2 | {t1,t2} | {} |
| black | 1965 | f | 2 | {t3,t4} | {} |
| black | 1964 | f | 2 | {t5,t6} | {} |
| white | 1964 | m | 3 | {t7,t9,t10} | {} |
| white | 1965 | f | 1 | {t8} | {} |
| white | 1967 | m | 2 | {t11,t12} | {} |

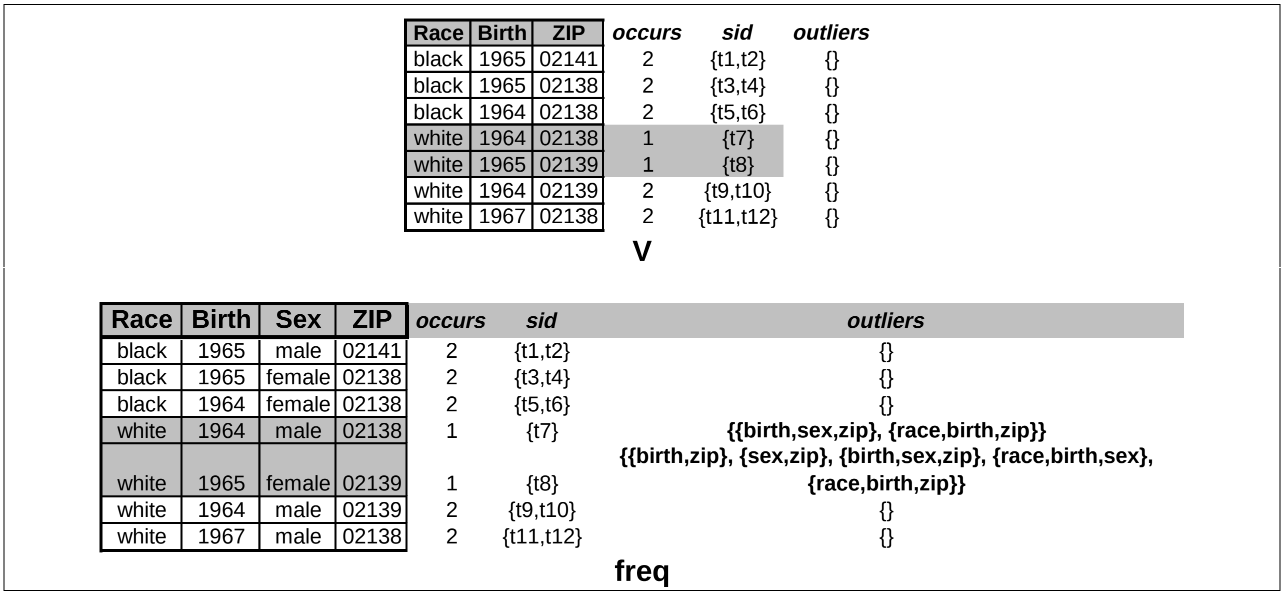
**V**

| Race | Birth | Sex | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|---|
| black | 1965 | male | 02141 | 2 | {t1,t2} | {} |
| black | 1965 | female | 02138 | 2 | {t3,t4} | {} |
| black | 1964 | female | 02138 | 2 | {t5,t6} | {} |
| white | 1964 | male | 02138 | 1 | {t7} | {{birth,sex,zip}} |
| white | 1965 | female | 02139 | 1 | {t8} | **{{birth,zip}, {sex,zip}, {birth,sex,zip}, {race,birth,sex}}** |
| white | 1964 | male | 02139 | 2 | {t9,t10} | {} |
| white | 1967 | male | 02138 | 2 | {t11,t12} | {} |

**freq**

**Figure 69 freq and V at *Most* x *More* x *Identifying* in *CombinationTest()*, step 6.1**

Figure 70 shows the frequency list V generated by *MarkOutliers()* at step 6.1 in *CombinationTest()* when it examines *Race* × *BirthDate* × *ZIP*. The combination where *Race*="white", *BirthDate*="1964" and *ZIP*="02138" occurs only once and appears in the tuple identified as *t7* in PT. Likewise, the combination where *Race*="white", *BirthDate*="1965" and *ZIP*="02139" occurs only once and appears in the tuple identified as *t8* in PT. As a result, *outliers* in freq is updated to include {*Race*, *BirthDate*, *ZIP*} for these elements. Depictions of the resulting V and freq tables are shown in Figure 70.



| Race | Birth | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|
| black | 1965 | 02141 | 2 | {t1,t2} | {} |
| black | 1965 | 02138 | 2 | {t3,t4} | {} |
| black | 1964 | 02138 | 2 | {t5,t6} | {} |
| white | 1964 | 02138 | 1 | {t7} | {} |
| white | 1965 | 02139 | 1 | {t8} | {} |
| white | 1964 | 02139 | 2 | {t9,t10} | {} |
| white | 1967 | 02138 | 2 | {t11,t12} | {} |

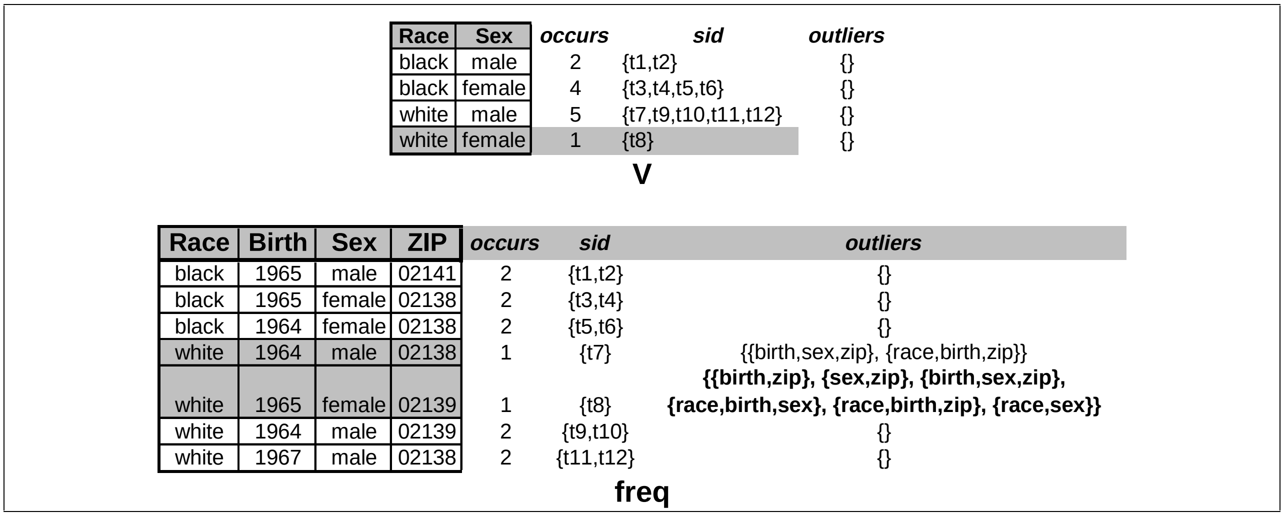
V

| Race | Birth | Sex | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|---|
| black | 1965 | male | 02141 | 2 | {t1,t2} | {} |
| black | 1965 | female | 02138 | 2 | {t3,t4} | {} |
| black | 1964 | female | 02138 | 2 | {t5,t6} | {} |
| white | 1964 | male | 02138 | 1 | {t7} | **{{birth,sex,zip}, {race,birth,zip}}** |
| white | 1965 | female | 02139 | 1 | {t8} | **{{birth,zip}, {sex,zip}, {birth,sex,zip}, {race,birth,sex}, {race,birth,zip}}** |
| white | 1964 | male | 02139 | 2 | {t9,t10} | {} |
| white | 1967 | male | 02138 | 2 | {t11,t12} | {} |

freq

**Figure 70 freq and V at *Most* x *More* x *Identifying* in *CombinationTest()*, step 6.1**

The next combinations examined result from *More* × *Identifying*. These are the specific combinations *Race* × *Sex* and *Race* × *ZIP*. Figure 71 shows the frequency list V generated by *MarkOutliers()* at step 6.2 in *CombinationTest()* when it examines *Race* × *Sex*. The combination where *Race*="white" and *Sex*="female" occurs only once and appears in the tuple identified as *t8* in PT. As a result, *outliers* in freq is updated to include {*Race*, *Sex*} for this element. Depictions of the resulting V and freq tables are shown in Figure 71.



| Race | Sex | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|
| black | male | 2 | {t1,t2} | {} |
| black | female | 4 | {t3,t4,t5,t6} | {} |
| white | male | 5 | {t7,t9,t10,t11,t12} | {} |
| white | female | 1 | {t8} | {} |

V

| Race | Birth | Sex | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|---|
| black | 1965 | male | 02141 | 2 | {t1,t2} | {} |
| black | 1965 | female | 02138 | 2 | {t3,t4} | {} |
| black | 1964 | female | 02138 | 2 | {t5,t6} | {} |
| white | 1964 | male | 02138 | 1 | {t7} | {{birth,sex,zip}, {race,birth,zip}} |
| white | 1965 | female | 02139 | 1 | {t8} | **{{birth,zip}, {sex,zip}, {birth,sex,zip}, {race,birth,sex}, {race,birth,zip}, {race,sex}}** |
| white | 1964 | male | 02139 | 2 | {t9,t10} | {} |
| white | 1967 | male | 02138 | 2 | {t11,t12} | {} |

freq

**Figure 71 freq and V at *More* x *Identifying* in *CombinationTest()*, step 6.2**

Figure 72 shows the frequency list V generated by *MarkOutliers()* at step 6.2 in *CombinationTest()* when it examines *Race* × *ZIP*. None of the combinations appear less than *k* times. As a result, freq is not modified.

| Race | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|
| black | 02141 | 2 | {t1,t2} | {} |
| black | 02138 | 4 | {t3,t4,t5,t6} | {} |
| white | 02138 | 3 | {t7,t11,t12} | {} |
| white | 02139 | 3 | {t8,t9,t10} | {} |

**Figure 72 V at *More* x *Identifying* in *CombinationTest()*, step 6.2**

The next combinations examined result from *Most* × *Identifying*. This is the specific combination *Race* × *BirthDate*. Figure 73 shows the frequency list V generated by *MarkOutliers()* at step 7.1 in *CombinationTest()* when it examines *Race* × *BirthDate*. The combination where *Race*="white" and *BirthDate*="1965" occurs only once and appears in the tuple identified as *t8* in PT. As a result, *outliers* in freq is updated to include {*Race*, *BirthDate*} for this element. Depictions of the resulting V and freq tables are shown in Figure 73.

| Race | Birth | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|
| black | 1965 | 4 | {t1,t2,t3,t4} | {} |
| black | 1964 | 2 | {t5,t6} | {} |
| white | 1964 | 3 | {t7,t9,t10} | {} |
| white | 1965 | 1 | {t8} | {} |
| white | 1967 | 2 | {t11,t12} | {} |

**V**

| Race | Birth | Sex | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|---|
| black | 1965 | male | 02141 | 2 | {t1,t2} | {} |
| black | 1965 | female | 02138 | 2 | {t3,t4} | {} |
| black | 1964 | female | 02138 | 2 | {t5,t6} | {} |
| white | 1964 | male | 02138 | 1 | {t7} | {{birth,sex,zip}, {race,birth,zip}} |
| white | 1965 | female | 02139 | 1 | {t8} | **{{birth,zip}, {sex,zip}, {birth,sex,zip}, {race,birth,sex}, {race,birth,zip}, {race,sex}, {race,birth}}** |
| white | 1964 | male | 02139 | 2 | {t9,t10} | {} |
| white | 1967 | male | 02138 | 2 | {t11,t12} | {} |

**freq**

**Figure 73 freq and V at *Most* x *Identifying* in *CombinationTest()*, step 7.1**

That concludes the examination of combinations of values that occurs at step 4 of the μ-Argus algorithm. The contents of *outliers* in freq are displayed and the data holder is solicited for an attribute to generalize or not. This example continues as option 1, in which no further generalization is selected and as option 2, in which *ZIP* is generalized.

**Example, continued with option 1**

In this option, the example continues with no further generalization is selected. So, execution of the μ-Argus algorithm proceeds to step 8.

Figure 74 shows freq at the start of *SuppressOutliers()*. The attributes within the *outliers* for an element in freq are examined. The value associated with the attribute occurring in the most number of members of *outliers* is suppressed. This process continues until all members of a value associated with *outliers* contain at least one suppressed value. In Figure 74 there are two elements of freq that have non-empty values for *outliers*. These are the elements associated with *t7* and *t8*.

| Race | Birth | Sex | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|---|
| black | 1965 | male | 02141 | 2 | {t1,t2} | {} |
| black | 1965 | female | 02138 | 2 | {t3,t4} | {} |
| black | 1964 | female | 02138 | 2 | {t5,t6} | {} |
| white | 1964 | male | 02138 | 1 | {t7} | {{**birth**,sex,zip},<br>{race,**birth**,zip}} |
| white | 1965 | female | 02139 | 1 | {t8} | {{**birth**,zip}, {**sex**,zip},<br>{**birth**,**sex**,zip},<br>{race,**birth**,**sex**},<br>{race,**birth**,zip}, {race,**sex**},<br>{race,**birth**}} |
| white | 1964 | male | 02139 | 2 | {t9,t10} | {} |
| white | 1967 | male | 02138 | 2 | {t11,t12} | {} |

**Figure 74 freq at *SuppressOutliers()* in μ-Argus algorithm, step 8**

The value of *outliers* for the element associated with *t7* is {{*BirthDate*, *Sex*, *ZIP*}, {*Race*, *BirthDate*, *ZIP*}}. The attributes *BirthDate* and *ZIP* occur most frequently, so either can be suppressed. *BirthDate* is selected. That means, the value associated with *BirthDate* for this element will be suppressed. At that time, the outlier combination *BirthDate* × *Sex* × *ZIP* and *Race* × *BirthDate* × *ZIP* will each contain a suppressed value, so no further suppression is needed for the element associated with *t7*.

The value of *outliers* for the element associated with *t8* is {{*BirthDate*, *ZIP*}, {*Sex*, *ZIP*}, {*BirthDate*, *Sex*, *ZIP*}, {*Race*, *BirthDate*, *Sex*}, {*Race*, *BirthDate*, *ZIP*}, {*Race*, *Sex*}, {*Race*, *BirthDate*}}. The attribute *BirthDate* occurs most frequently, so it will be suppressed. That means, the value associated with *BirthDate* for this element will be suppressed. At that time, the remaining outlier combinations associated with *t8* are {{*Sex*, *ZIP*}, {*Race*, *Sex*}}. The attribute

*Sex* now occurs most frequently, so it will also be suppressed. That means, the values associated with *BirthDate* and with *Sex* for this element will be suppressed. No further suppression is needed for the element associated with *t8*.

Figure 75 shows the final result from the μ-Argus algorithm, which is listed in Figure 50.

| id | Race | BirthDate | Gender | ZIP |
|---|---|---|---|---|
| t1 | black | 1965 | male | 02141 |
| t2 | black | 1965 | male | 02141 |
| t3 | black | 1965 | female | 02138 |
| t4 | black | 1965 | female | 02138 |
| t5 | black | 1964 | female | 02138 |
| t6 | black | 1964 | female | 02138 |
| t7 | white |  | male | 02138 |
| t8 | white |  |  | 02139 |
| t9 | white | 1964 | male | 02139 |
| t10 | white | 1964 | male | 02139 |
| t11 | white | 1967 | male | 02138 |
| t12 | white | 1967 | male | 02138 |

**MT**

**Figure 75 Result from μ-Argus algorithm listed in Figure 50**

Unfortunately, as was pointed out earlier, the actual μ-Argus algorithm does not exhaust all known outlying combinations of values when deciding on which values to suppress. Figure 76 shows the results when the private table PT along with the parameters specified in this example was provided to the actual μ-Argus program. Fewer cells are suppressed even though the combinations of values identified as outliers were the same.

| id | Race | BirthDate | Gender | ZIP |
|---|---|---|---|---|
| t1 | black | 1965 | male | 02141 |
| t2 | black | 1965 | male | 02141 |
| t3 | black | 1965 | female | 02138 |
| t4 | black | 1965 | female | 02138 |
| t5 | black | 1964 | female | 02138 |
| t6 | black | 1964 | female | 02138 |
| t7 | white | 1964 | male | 02138 |
| t8 | white |  | female | 02139 |
| t9 | white | 1964 | male | 02139 |
| t10 | white | 1964 | male | 02139 |
| t11 | white | 1967 | male | 02138 |
| t12 | white | 1967 | male | 02138 |

**MT actual**

**Figure 76 Actual result from the real μ-Argus program**

**Example, continued with option 2**

In this option, the example has proceeded to step 6 of the μ-Argus algorithm as before. In this option however, execution continues by assuming the data holder selects *ZIP* as the attribute to generalize, where as the previous option assumed no attributes were selected to generalize. The contents of freq before this decision is made are shown in Figure 73. The *generalize()* method is listed in Figure 55. It replaces the values associated with *ZIP* in freq with the values that appear one level up *ZIP*'s value generalization hierarchy, which is shown in Figure 33. The result is to replace the 5-digit ZIP values with their first 4-digits. Step 7 of the μ-Argus algorithm resets the values associated with *outliers* in freq to the empty set. The resulting contents of freq from theses steps are shown in Figure 77. The tuple identified as *t8* remains an outlier.

| Race | Birth | Sex | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|---|
| black | 1965 | male | 0214* | 2 | {t1,t2} | {} |
| black | 1965 | female | 0213* | 2 | {t3,t4} | {} |
| black | 1964 | female | 0213* | 2 | {t5,t6} | {} |
| white | 1965 | female | 0213* | 1 | {t8} | {} |
| white | 1964 | male | 0213* | 3 | {t9,t10, t7} | {} |
| white | 1967 | male | 0213* | 2 | {t11,t12} | {} |

**Figure 77 freq after generalize *ZIP***

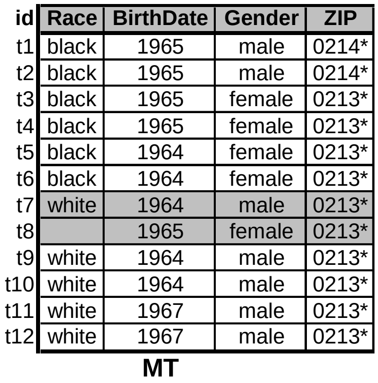
Execution of the μ-Argus continues by looping back to step 4. The method *CombinationTest()*, which is listed in Figure 56, computes 2- and 3-combinations across *Most*, *More* and *Identifying* to determine which combinations of values, if any, do not occur at least *k* times. In this case, only some combinations of values involving the tuple identified as *t8* do not adhere to the *k* requirement. The specific combinations are listed in the contents of freq shown in Figure 78.

| Race | Birth | Sex | ZIP | *occurs* | *sid* | *outliers* |
|---|---|---|---|---|---|---|
| black | 1965 | male | 0214* | 2 | {t1,t2} | {} |
| black | 1965 | female | 0213* | 2 | {t3,t4} | {} |
| black | 1964 | female | 0213* | 2 | {t5,t6} | {} |
| white | 1965 | female | 0213* | 1 | {t8} | {{**race**,birth,sex}, {**race**,birth,zip}, {**race**,sex}, {**race**,birth}} |
| white | 1964 | male | 0213* | 3 | {t9,t10, t7} | {} |
| white | 1967 | male | 0213* | 2 | {t11,t12} | {} |

**Figure 78 freq with *outliers* updated**

That concludes the examination of combinations of values that occurs at step 4 of the μ-Argus algorithm. The contents of *outliers* in freq are displayed and the data holder is solicited for an attribute to generalize or not. At this time, the data holder is assumed not to opt for further generalization. As a result, step 8 of the μ-Argus algorithm executes. The S*uppressOutliers()* method executes; it is listed in Figure 62.

The value of *outliers* for the element associated with *t8* is {{*Race*, *BirthDate*, *Sex*}, {*Race*, *BirthDate*, *ZIP*}, {*Race*, *Sex*}, {*Race*, *BirthDate*}}. The attribute *Race* occurs most frequently, so it will be suppressed. No further suppression is needed for the element associated with *t8* because all members of *outliers* now contain a suppressed value. The final table resulting from the μ-Argus algorithm based on the option of generalizing ZIP is shown in Figure 79.



| id | Race | BirthDate | Gender | ZIP |
|---|---|---|---|---|
| t1 | black | 1965 | male | 0214* |
| t2 | black | 1965 | male | 0214* |
| t3 | black | 1965 | female | 0213* |
| t4 | black | 1965 | female | 0213* |
| t5 | black | 1964 | female | 0213* |
| t6 | black | 1964 | female | 0213* |
| t7 | white | 1964 | male | 0213* |
| t8 |  | 1965 | female | 0213* |
| t9 | white | 1964 | male | 0213* |
| t10 | white | 1964 | male | 0213* |
| t11 | white | 1967 | male | 0213* |
| t12 | white | 1967 | male | 0213* |

**MT**

**Figure 79 Resulting table from μ-Argus algorithm with manual generalize *ZIP***

## 7.3 Comparison to Mingen

A comparison to MinGen [109] requires examining: (1) the computational complexity of the algorithm to ensure it operates in reasonable time; (2) the correctness of the algorithm in terms of *k*-anonymity protection; and, (3) whether the algorithm distorts minimally. These are discussed in the following subsections.

### 7.3.1 Complexity of the μ-Argus algorithm

The μ-Argus algorithm listed in Figure 50 with supporting methods in Figure 51 through Figure 62 was not written as efficiently as possible. Nevertheless, here is a walk through the algorithm noting the computational complexity of each part.

The *freqSetup()* method, which is listed in Figure 51, is executed in step 2. If the contents of PT is sorted over the attributes QI beforehand, then the determination of how many tuples in PT[QI] correspond to the same element in freq can be determined in $O(|\mathrm{PT}|\ log\ |\mathrm{PT}|)$ time. Otherwise, the construction of freq and the determination of the number of tuples in PT[QI] that correspond to an element in freq is performed in $O(|\mathrm{PT}|^2)$ time.

The sub-steps of step 3 of the μ-Argus algorithm operate |QI| times. On each iteration of these sub-steps, a frequency list is generated and generalization may be performed. The construction of a frequency list requires visiting each element of a frequency list and if changes are made due to generalization, the element is removed and then the modified element added. In order to avoid duplication of elements in a frequency list, all elements in the frequency list are compared to the element that is to be inserted. If the elements of freq were stored in a binary tree, then such a comparison could be done in $log(|\mathrm{freq}|)$ time. In the worst case, |freq| = |PT|; in all cases, |freq| ≤ |PT|. Step 3.3 can loop as much as $\sum_{i=1}^{|\mathrm{QT}|}\left|\mathrm{DGH}_{\mathrm{Ai}}\right|$ times in its worst case, which requires each attribute to generalize one step at a time to its maximal element. Because $|\mathrm{DGH}_{\mathrm{Ai}}| << |\mathrm{PT}|$ in almost all cases, this term is dropped. In the listing of the *freqConstruct()* and *generalize()* methods provided, the contents of freq are not stored in a binary tree and so the computation, in the worst case is, $O(|\mathrm{PT}|^2)$ time. Because this process is done on each iteration, the computational time for step 3 of the μ-Argus algorithm is $O(\ |\mathrm{QI}| \bullet |\mathrm{PT}|\ log\ |\mathrm{PT}|)$, if freq was stored as a binary tree, or $O(\ |\mathrm{QI}| \bullet |\mathrm{PT}|^2)$ as the methods are written.

Steps 4 through 7 of the μ-Argus algorithm perform a loop of reviewing 2- and 3- combinations, displaying them, and possibly generalizing an attribute. This loop is executed one or more times, depending on the data holder. The number of iterations is not likely to be large, so in this computation I will consider it a negligible constant.

Step 4 of the μ-Argus algorithm executes the *CombinationTest()* method. The goal of this method is to generate some 2- and 3-combinations and then determine which, if any, adhere to the *k* requirement. The number of 2- combinations, assuming all such combinations within QI are to be examined, would be $\frac{|\mathrm{QI}|!}{2(|\mathrm{QI}|-2)!}$ and 3-combinations would be $\frac{|\mathrm{QI}|!}{6(|\mathrm{QI}|-3)!}$. These are roughly characterized as $O(|\mathrm{QI}|^2)$. With the constructions of frequency lists included, the computational time for this step is $O(\ |\mathrm{QI}|^2 \bullet |\mathrm{PT}|\ log\ |\mathrm{PT}|)$, if freq was stored as a binary tree, or $O(\ |\mathrm{QI}|^2 \bullet |\mathrm{PT}|^2)$ as the methods are written.

Step 5 is a walk through each element of freq reporting the value of *outliers* for that element. That executes in $|freq|$ time. In the worst case $|freq| = |PT|$, so this step executes in $O(|PT|)$ time. Each iteration of the loop in step 6 of the μ-Argus algorithm, if executed at all, executes in is $O(\ |PT|\ log\ |PT|)$, if freq was stored as a binary tree, or $O(\ |PT|^2)$ as the methods are written. Step 7, like step 5 is a walk through each element of freq and so, it executes in $|freq|$ time. In the worst case $|freq| = |PT|$, so step 7 executes in $O(|PT|)$ time.

The S*uppressOutliers()* method in step 8 of the μ-Argus algorithm has an outer loop that visits each element of freq, and within the outer loop are inner loops based on the contents of *outliers* for that element. In the worst case, $|freq| = |PT|$ and $|outliers|$ is nearly $|QI|^2$. The method *freqCleanup()* executes in $O(\ |PT|\ log\ |PT|)$, if freq was stored as a binary tree, or $O(\ |PT|^2)$ otherwise. So, the computation of the method is $O(|QI|^3 \bullet |PT| + |PT|\ log\ |PT|)$ if freq is stored as a binary tree or $O(|QI|^3 \bullet |PT| + |PT|^2)$ otherwise.

Step 9 of the μ-Argus algorithm executes the *reconstruct()* method, which visits each element of freq and generates tuple(s) for MT based on the element. This method executes in $|freq|$ time, which is $O(|PT|)$.

Finally, the overall computational complexity of the μ-Argus algorithm listed in Figure 50 is characterized by $O(|QI|^3 \bullet |PT| + |PT|\ log\ |PT|)$ if freq is stored as a binary tree or $O(|QI|^3 \bullet |PT| + |PT|^2)$ otherwise. In most databases, $|QI| << |PT|$. So, the overall complexity for the μ-Argus algorithm is $O(|PT|\ log\ |PT|)$ if freq is stored as a binary tree or $O(|PT|^2)$ otherwise. In comparison to the computational complexity of MinGen [110] and Equation 1 (on page 87), the computational complexity of the μ-Argus algorithm is practical and extremely fast.

### 7.3.2 Correctness of the μ-Argus algorithm

The correctness of the μ-Argus algorithm relies on its ability to produce solutions that adhere to a given *k*-anonymity requirement, assuming of course a proper quasi-identifier and a proper value for *k* have been provided. In this subsection, I will show that the μ-Argus algorithm provides solutions that do not necessarily adhere to a given *k*-anonymity requirement. As a result, tables generated by μ-Argus may not provide adequate protection. Here is a walk through the program, noting correctness problems.

After step 3 of the μ-Argus algorithm listed in Figure 50 concludes, each value associated with each attribute is guaranteed to appear at least *k* times. While this is a necessary condition to satisfy the *k* requirement, it is not itself sufficient to ensure that combinations of values also adhere to the *k* requirement. This note is not a claim of an error in correctness as much as a clarification that step 3 does not itself guarantee adherence to the *k* requirement.

**Example.**

Consider the private table PT shown in Figure 34 with a quasi-identifier QI = {*Race*, *Gender*} and a *k*-anonymity requirement of *k*=2. Each value associated with *Race* and each value associated with *Gender* appears more k times, but in combination ["white", "female"] occurs only once.

In order to make sure combinations of values adhere to the *k* requirement, values must be examined in combination. Step 4 of the μ-Argus algorithm executes the *CombinationTest()* method to examine combinations of values. Unfortunately, not all possible combinations across the quasi-identifier are examined. Only some 2- and 3- combinations are examined. There may be 4-combinations or beyond that are unique and not examined and there may be 2- or 3-combinations not examined at all. As a result, the μ-Argus algorithm at this step cannot guarantee that all combinations of values adhere to the *k* requirement.

**Example.**

Consider the private table PT shown in Figure 34 with a quasi-identifier QI = {*Race*, *BirthDate*, *Gender*, *ZIP*}, where *Most* = {*BirthDate*}, *More* = {*Gender*, *ZIP*} and *Identifying* = {*Race*} and a *k*-anonymity requirement of *k*=2. The actual μ-Argus program provides the table MTactual shown in Figure 76 as a generalization of PT over QI. Notice however that the tuple identified as *t7* is unique over QI. It contains the unique occurring 4-combination ["white", "1964", "male", "02138"]. Therefore, MTactual does not satisfy the *k* requirement.

Only election by the data holder to generalize an attribute in step 6 of the μ-Argus algorithm and the automatic suppression of values done by the *SuppressOutliers()* method in step 8 of the μ-Argus algorithm are ways to further distort data after step 3. Unfortunately, neither of these steps ensures that combinations of values adhere to the *k* requirement. Actions taken by theses steps do not necessarily enforce the *k* requirement.

A data holder's decision to generalize or not is made before the results of suppression are determined. Yet, the responsibility of adhering to the *k* requirement is passed to the data holder, who must specify whether further generalization is needed, and if so, which attribute(s) to generalize. These decisions are made with limited and indirect information from the μ-Argus algorithm.

**Example.**

Consider the private table PT shown in Figure 34 with a quasi-identifier QI = {*Race*, *BirthDate*, *Gender*, *ZIP*}, where *Most* = {*BirthDate*}, *More* = {*Gender*, *ZIP*} and *Identifying* = {*Race*} and a *k*-anonymity requirement of *k*=2. Figure 75 shows the result from the μ-Argus algorithm with no additional generalization elected. There is no recommendation as to whether an attribute should be generalized and if so, which one(s). Figure 79 shows the results from the μ-Argus algorithm after values associated with *ZIP* were generalized. There is no preference posed by the algorithm for one solution over another even though one is more distorted than the other and because of the uniqueness of suppressed values, neither solution adheres to the *k*-anonymity requirement.

The data holder may incorrectly believe that the suppression process in step 8 will ensure adequate protection, because the μ-Argus algorithm performs suppression automatically after generalization decisions by the data holder conclude. But the *SuppressOutliers()* method is problematic. Some combinations of values whose attributes are identified in *outliers* may not have values suppressed values in the resulting table even though all combinations reported in *outliers* is known to not adhere to the *k* requirement. This is obviously a problem with the real μ-Argus implementation.

**Example.**

Consider the private table PT shown in Figure 34 with a quasi-identifier QI = {*Race*, *BirthDate*, *Gender*, *ZIP*}, where *Most* = {*BirthDate*}, *More* = {*Gender*, *ZIP*} and *Identifying* = {*Race*} and a *k*-anonymity requirement of *k*=2. Figure 76 shows the actual result from the real μ-Argus program. In comparison, Figure 75 shows the result from the μ-Argus algorithm. Notice that in the actual result, values related to the tuple identified as *t7* are not suppressed even though *CombinationTest()* identified {*BirthDate*, *Sex*, *ZIP*} and {*Race*, *BirthDate*, *ZIP*} as combinations that had values within *t7* that did not adhere to the *k* requirement; see Figure 74.

### 7.3.3 Summary data attack on μ-Argus results

μ-Argus does not enforce the *k*-anonymity requirement on suppressed values. As a result, tables released from μ-Argus can be vulnerable to inference attacks based on summary data. If the frequencies of values contained within the privately held information are released separately for each attribute, which is often the case in statistical reports and summary data, then this information can be used to infer suppressed values if the suppressed values themselves do not adhere to the *k*-anonymity requirement imposed on the other released values.

**Example.**

Summary data for the privately held information PT in Figure 34 is shown in Figure 46. Given a quasi-identifier QI = {*Race*, *BirthDate*, *Gender*, *ZIP*}, where *Most* = {*BirthDate*}, *More* = {*Gender*, *ZIP*} and *Identifying* = {*Race*} and a *k*-anonymity requirement of *k*=2, table MT in Figure 79 results from executing the μ-Argus algorithm on PT with QI and *k*. In this case, values associated with *ZIP* were generalized. Except for suppressed values, *k*-anonymity is satisfied for all other tuples. However, using the summary data, the missing or suppressed values can be inferred exactly. To combat this problem, the *k*-anonymity requirement must be satisfied on all values, including suppressed ones. Figure 80 shows a generalization of MT in which *k*-anonymity is also enforced on suppressed values. As you can see, the summary information does not allow one to confidently infer the suppressed values.

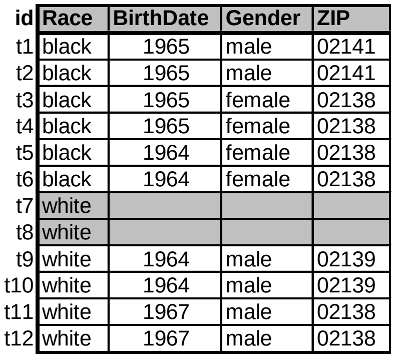
| id | Race | BirthDate | Gender | ZIP |
|---|---|---|---|---|
| t1 | black | 1965 | male | 0214* |
| t2 | black | 1965 | male | 0214* |
| t3 |  | 1965 | female | 0213* |
| t4 |  | 1965 | female | 0213* |
| t5 | black | 1964 | female | 0213* |
| t6 | black | 1964 | female | 0213* |
| t7 | white | 1964 | male | 0213* |
| t8 |  | 1965 | female | 0213* |
| t9 | white | 1964 | male | 0213* |
| t10 | white | 1964 | male | 0213* |
| t11 | white | 1967 | male | 0213* |
| t12 | white | 1967 | male | 0213* |

**Figure 80 Table from μ-Argus algorithm (Figure 79) with complementary suppression added**

It is important to realize that avoidance of a summary data attack is not wholly resolved by merely providing *k* indistinguishable tuples containing suppressed values. Inferences about the suppressions must not be further distinguished by the non-suppressed values. Within the *k*-anonymity framework, probabilistic attacks on distorted values are not necessarily resolved.

**Example.**

Summary data for the privately held information PT in Figure 34 is shown in Figure 46. Given a quasi-identifier QI = {*Race*, *BirthDate*, *Gender*, *ZIP*}, where *Most* = {*BirthDate*}, *More* = {*Gender*, *ZIP*} and *Identifying* = {*Race*} and a *k*-anonymity requirement of *k*=2, table MT in Figure 75 results from executing the μ-Argus algorithm on PT with QI and *k*. Except for suppressed values, *k*-anonymity is satisfied for all other tuples. However, using the summary data, the missing or suppressed values can be inferred exactly. To combat this problem, the *k*-anonymity requirement must be satisfied on all values, including suppressed ones. Figure 81 shows a generalization of MT in which *k*-anonymity is also enforced on suppressed values. However, the summary data informs that one of the suppressed tuples pertains to a "male" and the other a "female". If the non-suppressed values that are associated with these tuples in PT were gender specific, then values for gender could be confidently inferred and the *k* requirement would no longer be valid.



| id | Race | BirthDate | Gender | ZIP |
|---|---|---|---|---|
| t1 | black | 1965 | male | 02141 |
| t2 | black | 1965 | male | 02141 |
| t3 | black | 1965 | female | 02138 |
| t4 | black | 1965 | female | 02138 |
| t5 | black | 1964 | female | 02138 |
| t6 | black | 1964 | female | 02138 |
| t7 | white | | | |
| t8 | white | | | |
| t9 | white | 1964 | male | 02139 |
| t10 | white | 1964 | male | 02139 |
| t11 | white | 1967 | male | 02138 |
| t12 | white | 1967 | male | 02138 |

**Figure 81 Table from μ-Argus algorithm (Figure 75) with complementary suppression added**

### 7.3.4 Distortion and the μ-Argus algorithm

In terms of assessing the quality of generalized data that adhere to a *k*-anonymity requirement, it is important to note whether: (1) the resulting data are minimally generalized – i.e., not a generalization of another generalization that satisfies the same *k*-anonymity requirement; and, (2) the data are minimally distorted – i.e., of all minimal generalizations that satisfy the *k*-anonymity requirement, none have more precision retained in the data. In this subsection I will show that the μ-Argus algorithm does not necessarily provide minimally generalized solutions or minimally distorted ones, even in cases where its solutions do adhere to a *k*-anonymity requirement.

On the one hand, μ-Argus makes crude decisions – generalizing all values associated with an attribute. On the other hand, μ-Argus suppresses values at the cell level. Algorithms that make all decisions at the cell-level can potentially provide optimal results.

**Example.**

Given the privately held information PT in Figure 34, the Figure 79 provides the table MT, where μ-Argus(PT) = MT for *k*=2, quasi-identifier QI={*Race*, *BirthDate*, *Gender*, *ZIP*}, where *Most* = {*BirthDate*}, *More* = {*Gender*, *ZIP*} and *Identifying* = {*Race*}, and $\forall i=1,\ldots,|QI|$, $DGH_{Ai}$ are domain generalization hierarchies based on the depictions in Figure 33. The Figure 80 provides table MGT, where MGT is MT with complementary suppression added. MT does not adhere to the *k*-anonymity requirement; MGT does. The precision, *Prec*(MGT) with respect to $DGH_{Ai}$ is 0.754. In comparison, Figure 35 provides GT1, where MinGen(PT)=GT1. It is a *k*-minimal distortion of PT over QI with respect to $DGH_{Ai}$ where *Prec*(GT1)=0.83. The MinGen result therefore has less distortion based on cell-level generalization and suppression. Notice that although *Prec*(MT)=0.85, MT does not adhere to the *k*-anonymity requirement.

Another problem is the data holder's unrestricted and mostly unguided selection of which attribute, if any, to generalize. There is no recommendation made or sufficient metrics provided for the data holder to make an informed decision. The μ-Argus algorithm makes the assumption that the data holder knows best, which is reasonable only if sufficient information about the ramifications to protection and distortion are provided to the data holder about such decisions at the time the data holder must decide. This is especially important because the subject data at that time reside in such an intermediate state that the resulting consequences are not necessarily clear. The absence of this information allows the data holder to guide the μ-Argus program into providing results that are more or less generalized than needed.

**Example.**

Given the privately held information PT in Figure 34, the Figure 75 and the Figure 79 provide versions the tables MT1 and MT2, respectively, where μ-Argus(PT) = MT1 and μ-Argus(PT) = MT2 for *k*=2, quasi-identifier QI={*Race*, *BirthDate*, *Gender*, *ZIP*}, where *Most* = {*BirthDate*}, *More* = {*Gender*, *ZIP*} and *Identifying* = {*Race*}, and ∀*i*=1,…,|QI|, $DGH_{Ai}$ are domain generalization hierarchies based on the depictions in Figure 33. Table MT1 has values associated with *ZIP* generalized, as directed by the data holder. The Figure 80 provides table MGT1, where MGT1 is MT1 with complementary suppression added. Likewise, The Figure 81 provides table MGT2, where MGT2 is MT2 with complementary suppression added. Neither MT1 nor MT2 adhere to the *k*-anonymity requirement; but MGT1 and MGT2 do. The precision, *Prec*(MGT1) with respect to $DGH_{Ai}$ is 0.754. The precision, *Prec*(MGT2) with respect to $DGH_{Ai}$ is 0.792. So, MGT1 does more distortion than is necessary. The data holder made the decision to generalize the values of *ZIP* with only the information provided in Figure 74. At that time, it is not clear that MGT1 would be more distorting and further, it is not clear that selecting another attribute other than *ZIP* to generalize would not reveal better results.

A third problem is the selection of values to suppress. After some values may have been generalized, combinations of 2 and 3 values that do not occur at least *k* times in the data are identified in the μ-Argus algorithm. As stated earlier, these are termed outliers. At least one value in each outlier combination is to be suppressed. Even though the actual μ-Argus algorithm identifies all such combinations, it does not suppress a value from each combination, and so, it can leave data vulnerable. See Figure 74 and Figure 76 versus Figure 74 and Figure 75 for an example.

## 7.4 Comparison to Datafly

I will briefly compare the results of these two systems. In the Datafly System, generalizing across a quasi-identifier ensures that the corresponding tuples will adhere to the *k* requirement. The μ-Argus program however, only checks some 2- or 3- combinations; there may exist unique combinations across 4 or more attributes that would not be detected. Treating a quasi-identifier as a single attribute that must adhere to the *k* requirement, as done in the Datafly System provides more secure releases of data. Further, since the number of attributes, especially demographic attributes, in a health database is large, this strategy of examining only some 2- and 3-combinations may prove to be a serious handicap when using the μ-Argus system with health data.

While both μ-Argus and Datafly employ attribute-level generalization, μ-Argus employs cell-level suppression where Datafly suppresses at the tuple level. Therefore, the granularity of distortion is better with μ-Argus. Results produced by μ-Argus can be less distorting than with Datafly, even when both adhere to k-anonymity.

## 7.5 μ-Argus as an anonymous data system

μ-Argus uses the following disclosure limitation techniques: de-identification, generalization, and suppression. Below is a description of the framework in which μ-Argus operates.

S = {*subjects whose information is included in PT*}

P = *set of all people whose information could possibly be in PT*

PT = *privately held information about S*

*QI = set of attributes with replications in E*

U = P

MT = *μ-Argus* (PT)

E = set of publicly available information in today's society

G = set of standard communication methods.

*f = μ-Argus algorithm*

The system **A**(S, P, PT, *QI*, U, {MT}, E, G, *μ-Argus*) is not an $\mathbf{ADS_0}$.

Informal proof.
Let PT be data in Figure 34.
There can exist fewer than *k* tuples in MT having the same values across *QI*,
as shown in Figure 75 and Figure 76.
So, *k*-map protection is not provided and **A** is not an $\mathbf{ADS_0}$.

## 7.6 Future work

1. One could view the contents for the frequency list used in both the Datafly algorithm and the μ-Argus algorithm as a matrix. Doing so, allows one to explore linear algebra techniques as ways to identify outliers by likening the frequencies to coefficients in a system of simultaneous equations. Some progress along these lines has resulted from linear programming approaches that utilize cell suppression [111]. Explore the use and deployment of linear algebra techniques as solutions to these kinds of problems.

2. The μ-Argus algorithm, which is listed in Figure 50, can be completely automated to work without data holder intervention and also made to adhere to *k*-anonymity while distorting the data as minimally as possible given the application of generalization enforced at the attribute level and suppression enforced at the cell level. Modify the algorithm along these lines to construct an Optimal μ-Argus algorithm and report on its computational complexity and correctness.

3. Prove that a solution based on the μ-Argus approach must examine all combinations of values within the quasi-identifier. Or, show where tradeoffs are possible to examine fewer combinations of values.

4. The μ-Argus algorithm presented in Figure 50 was not written to be as computationally efficient as possible. Examine this algorithm and its supporting algorithms and improve the computational complexity or prove the minimum complexity required for this approach. Examine and describe best case, worst case and general case scenarios.

5. The *SuppressOutliers()* algorithm, which is listed in Figure 62, selects values to be suppressed from each combination of values known to be an outlier in a tuple. The algorithm selects the value within the tuple that occurs the most often in all combinations identified as outliers. The strategy of selecting the most frequent value is done repeatedly on the values of a tuple until each combination of values identified as being an outlier contains at least one value that is suppressed. This approach may not necessarily provide the least distorting results. That is, there may exists situations in which the heuristic of suppressing the most frequently occurring value in this situation leads to unnecessary suppression. Prove whether this heuristic always provides a minimal number of suppressed values; and if not explore other strategies or algorithms that provide a minimal number of suppressed values. Set covering techniques may be useful.

**Example**

Given the privately held information PT in Figure 34, the Figure 76 provides the table MT, where μ-Argus(PT) = MT for $k$=2, quasi-identifier QI={*Race*, *BirthDate*, *Gender*, *ZIP*}, where *Most* = {*BirthDate*}, *More* = {*Gender*, *ZIP*} and *Identifying* = {*Race*}, and $\forall i$=1,…,|QI|, $DGH_{Ai}$ are domain generalization hierarchies based on the depictions in Figure 33. Figure 75 shows the intermediate sate of the data including outliers before *SuppressOutliers()* executes. The outliers for the tuples identified as *t8* are shown in Figure 82. Each outlier combination appears as a row. Each attribute aligns vertically. The attribute for *Birth* (for *BirthDate*) appears most often (5 times). It is suppressed, leaving the combinations {*zip*, *sex*} and {*sex*, *race*} as outlier combinations with no suppressed value. Of these attributes, *sex* (for *Gender*) appears most often. So it is suppressed. Therefore, for tuple *t8* the values associated with *BirthDate* and *Gender* are suppressed, as shown in Figure 75.

| | | | |
|---|---|---|---|
| **birth** | zip | | |
| | zip | **sex** | |
| **birth** | zip | **sex** | |
| **birth** | | **sex** | race |
| **birth** | zip | | race |
| | | **sex** | race |
| **birth** | | | race |

**Figure 82 Combinations of attributes containing outliers**

However, Figure 83 shows the same outlier combinations as those in Figure 82 but with *zip* and *race* selected for suppression. Both the solution posed in Figure 82, which suppresses the values associated with *birth* and *sex*, and Figure 83, which suppresses the values associated with *zip* and *race*, provide the same amount of distortion when applied to *t8* because both solutions suppress two values. Both solutions also provide the same protection in that each outlier combination for *t8* has at least one value suppressed.

| | | | |
|---|---|---|---|
| birth | **zip** | | |
| | **zip** | sex | |
| birth | **zip** | sex | |
| birth | | sex | **race** |
| birth | **zip** | | **race** |
| | | sex | **race** |
| birth | | | **race** |

**Figure 83 Combinations of attributes containing outliers**

6. Implement a version of the μ-Argus approach using suppression as the only disclosure limitation technique employed. The *CombinationTest()* algorithm, which is listed in Figure 56, and the *SuppressOutliers()* algorithm, which is listed in Figure 62, form the basis for this revised approach. Once the revision is working, assess it computational complexity, correctness and data distortion. Then, revise the approach further to get results that are correct with minimal distortion. (This is related to #5 above.)

7. Improve the μ-Argus algorithm by providing complementary suppression so that resulting tables are not vulnerable to summary attacks. This involves enforcing the *k* requirement on suppressed values.

# Chapter 8 Results: *k*-Similar

In Chapter 6, the Datafly System was shown to sometimes over distort data. In Chapter 7, the μ-Argus System was shown to sometimes fail to provide adequate protection. In this chapter, I present my *k*-Similar algorithm, which uses generalization and suppression to find optimal solutions such that data are minimally distorted while still being adequately protected. Decisions are automatically made at the cell level that adhere to a given *k*-anonymity requirement [112] and that maximize the precision metric [113]. The *k*-similar algorithm achieves these goals by looking at the computational disclosure control problem as one of data clustering. In the well-known *k*-nearest neighbor or *k*-cluster algorithm [114], for example, data are partitioned into *k* groups based on minimizing a distance between tuples. In contrast, my *k*-similar algorithm divides data into groups such that the size of each group consists of *k* or more of the "closest" tuples; in this case, closeness is based on a minimal distance measure derived from distance vectors [115].

## 8.1 Overview of the *k*-Similar algorithm

More generally, the *k-similar* algorithm provides a solution to finding similarity matches in high dimensional space with data consisting of primarily categorical values. In this setting, traditional mining approaches have faced tremendous difficulty primarily because of troubles measuring "distance" between categorical values. The *k-similar* approach is based on combining generalization and suppression and on using the resulting hierarchies as a semantically useful grouping that reflects a partial ordering on values. By cell generalization, I mean that a value can be replaced by a less precise but semantically consistent alternative. Cell suppression in this context is considered the most general value possible because semantically no information is released. The distance between two values can then be measured in terms of the minimal level up the generalization hierarchy at which the two values have a common ancestor. This precision metric provides the basis for a semantically meaningful measure of distance [116]. Given a table and a value for *k*, the *k-similar* algorithm groups the tuples of the table in as many clusters as necessary such that each cluster contains at least *k* of its closest tuples. In terms of anonymity, having *k* tuples that are indistinguishable is the basis for *k*-anonymity protection.

## 8.2 Abstract of the *k*-Similar algorithm

The *k*-Similar algorithm is not a complete system like Datafly or μ-Argus. It is intended to fit within a system, such as Datafly's, replacing the core operational algorithm found there with the *k*-Similar algorithm. (A description of the overall Datafly System is provided on page 107.) Here is a summary of the setting in which the *k*-Similar algorithm operates.

Using the Datafly System as a shell for the *k*-Similar algorithm, the data holder provides an overall anonymity level (*A*), which is a value between 0 and 1. The data holder also provides a profile of the recipient by providing a linking likelihood ($P_f$) for each attribute that is also a value between 0 and 1. Based on these values an overall value for *k* is computed and quasi-identifier(s) are determined. For example, subsets of attributes where $P_f$=1 are treated as one concatenated attribute, or quasi-identifier, which must satisfy a *k*-anonymity requirement. Each attribute has a replacement algorithm that either uses equivalence class substitution, such as SSNs, or generalization based on a domain generalization hierarchy specific to that attribute. In summary, the *k*-Similar algorithm merely replaces the core Datafly algorithm within the system. The *k*-Similar algorithm therefore works with a quasi-identifier and a *k*-anonymity requirement that is to be enforced on the quasi-identifier. For convenience, I consider all attributes of the quasi-identifier as having equal weights (specifically, $P_f$=1 for each attribute of the quasi-identifier though a weighted precision metric has been provided [117]); and, I address only generalizable attributes of the quasi-identifier in isolation, ignoring those that would utilize equivalence class substitution.

Before I introduce the *k*-Similar algorithm itself, let me first expand the earlier discussion on distance vectors [118].

### 8.2.1 Distance vectors expanded

The *k-similar* algorithm uses generalization with suppression to group the closest *k* or more tuples together into clusters. Closeness between tuples can be determined in terms of the value generalization hierarchies [119] for the attributes. Basically, the distance between values is the level of the generalization hierarchy at which the values have the same ancestor.

## Definition. distance between values

Let $A$ be an attribute, $v_1$ and $v_2$ be values associated with $A$, and $f_i \in \mathrm{DGH}_A$, for $i$=1,…,$h$. The distance between the values $v_1$ and $v_2$ is the smallest $h$ for which $f_1(\ldots f_h(v_1)\ldots) = f_1(\ldots f_h(v_2)\ldots)$.

Given the definition above, the distance between values is the length of the shortest path from the ground domain to the domain in $\mathrm{DGH}_A$ in which both values share the same generalized value. By extension, the distance between two tuples can be expressed as a vector denoting the distance between values for each attribute. This is presented in the following definition of a distance vector.

## Definition. distance vector with respect to tuples

Let $t_i[A_1,\ldots,A_n]$ and $t_j[A_1,\ldots,A_n]$ be two tuples. The distance vector of $t_i$ to $t_j$ is the vector $\mathrm{DV}_{i,j}$ = $[d_1,\ldots,d_n]$ where each $d_z$, where $z$=1,…,$n$, is the distance between $t_i[A_z]$ and $t_j[A_z]$.

The relationship between the minimal generalization of a table and the distance vectors between tuples forms the basis for understanding the *k-similar* algorithm.

**Example**

Given the privately held information PT in Figure 84, the Figure 85 shows the distance vectors between every two tuples in PT. The quasi-identifier is QI={*HomeZIP*, *HospitalZIP*, *WorkZIP*} and $\forall i$=1,…,|QI|, $\mathrm{DGH}_{Ai}$ and $\mathrm{VGH}_{Ai}$ are the domain and value generalization hierarchies $\mathrm{DGH}_{ZIP}$ and $\mathrm{VGH}_{ZIP}$ based on the depiction in Figure 33. As shown in Figure 84, the distance vector of *t1* to *t2* is $\mathrm{DV}_{t1,t2}$ = [0,1,0] because *t1*[*HomeZIP*] is the same value as *t2*[*HomeZIP*], and *t1*[*WorkZIP*] is the same value as *t2*[*WorkZIP*], but *t1*[*HospitalZIP*] is NOT the same value as *t2*[*HospitalZIP*]. They can become the same value if they were generalized 1 level up $\mathrm{VGH}_{ZIP}$.

Likewise, the distance vector of *t1* to *t3* is $\mathrm{DV}_{t1,t3}$ = [0,0,2] because *t1*[*HomeZIP*] is the same value as *t3*[*HomeZIP*], and *t1*[*HospitalZIP*] is the same value as *t3*[*HospitalZIP*], but *t1*[*WorkZIP*] can become the same value as *t3*[*WorkZIP*] if they were generalized 2 levels up $\mathrm{VGH}_{ZIP}$. Similarly, the distance vector of *t1* to *t4* is $\mathrm{DV}_{t1,t4}$ = [0,1,1]; the distance vector of *t2* to *t3* is $\mathrm{DV}_{t2,t3}$ = [0,1,2]; the distance vector of *t2* to *t4* is $\mathrm{DV}_{t2,t4}$ = [0,0,1]; and, the distance vector of *t3* to *t4* is $\mathrm{DV}_{t3,t4}$ = [0,1,2].

| | *A1* | *A2* | *A3* |
|---|---|---|---|
| | **Home ZIP** | **Hospital ZIP** | **Work ZIP** |
| *t1* | 02138 | 02138 | 02138 |
| *t2* | 02138 | 02139 | 02138 |
| *t3* | 02138 | 02138 | 02141 |
| *t4* | 02138 | 02139 | 02139 |

**Figure 84 Private Table PT**

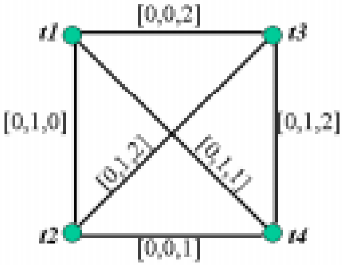


**Figure 85 Clique showing distance vectors between tuples of Figure 84**

To interpret distance vectors, the *k-similar* algorithm uses a distance function **dist** based on the precision metric *Prec()* and therefore, is typically defined as the sum of the normalized value of each element in the vector [120]. That is, given a vector $V_{x,y}$= [$d_1$, ..., $d_n$] associated with attributes, {$A_1$,…,$A_n$}:

$$dist(V_{x,y}) = \sum_{i=1}^{n} \frac{d_i}{|\mathrm{DGH}_{Ai}|}$$

Other distance functions correspond to different precision metrics. For example, using a weighted precision metric [121] would warrant the use of a corresponding weighted distance function. However, the distance function used must not only relate to the precision metric used but must also satisfy the properties of Euclidean geometry listed in Figure 86 for all possible tuples *x*, *y*, and *z*.

*(1)* **dist**($V_{x,y}$) $\geq 0$;
*(2)* **dist**($V_{x,y}$) = 0 iff $x = y$;
*(3)* **dist**($V_{x,y}$) = **dist**($V_{y,x}$); and
*(4)* **dist**($V_{x,y}$) $\leq$ (**dist**($V_{x,z}$) + **dist**($V_{z,y}$)).

**Figure 86 Euclidean Properties of distance function**

Lastly, Figure 87 contains operations and relations on distance vectors that determine a partial ordering on distance vectors and that determine containment. These are used by the *k-similar* algorithm, which is presented later in this subsection. In each of these cases, let $V_{x,y}$= [$d_{y1}$, ..., $d_{yn}$] and $V_{x,z}$= [$d_{z1}$, ..., $d_{zn}$] be distance vectors between tuples.

(1) $V_{xy} < V_{xz}$ iff $d_{yi} < d_{zi}$ for all $i$=1,…,$n$.
(2) $V_{xy} = V_{xz}$ iff $d_{yi} = d_{zi}$ for all $i$=1,…,$n$
(3) $V_{xy} \Theta V_{xz}$ = [min($d_{y1}$, $d_{z1}$), …, min($d_{yn}$, $d_{zn}$)]
(4) $V_{xy} \oplus V_{xz}$ = [max($d_{y1}$, $d_{z1}$), …, max($d_{yn}$, $d_{zn}$)]

**Figure 87 Relations on distance vectors**

## Definition. maximal distance vector

Given a table T[QI] and a set of tuples {$t_1$[QI], …, $t_m$[QI]} where for $i$=1,…,$m$, $t_i$∈T and $m \geq 2$, the maximal distance vector across the set of tuples is $V_{tjt1} \oplus \mathrm{V}_{tjt2} \oplus \ldots \oplus \mathrm{V}_{tjtm}$,where $j$ is 1, 2, …, or $m$.

The maximal distance vector across a set of tuples in a table is the distance vector that reports for each attribute, the level up the value generalization hierarchy for that attribute, at which all values associated with that attribute in the set of tuples is the same. It is computed by iteratively applying the ⊕ operator, defined in Figure 87, to all distances of one tuple in the set $t_j$ to all the other tuples. The result is the maximum level up the value generalization hierarchy that the tuple $t_j$ must combine with the other tuples.

**Example**

Given the privately held information PT in Figure 84, the Figure 85 shows the distance vectors between every two tuples in PT. The quasi-identifier is QI={*HomeZIP*, *HospitalZIP*, *WorkZIP*} and ∀$i$=1,…,|QI|, $\mathrm{DGH}_{\mathrm{Ai}}$ and $\mathrm{VGH}_{\mathrm{Ai}}$ are the domain and value generalization hierarchies $\mathrm{DGH}_{\mathrm{ZIP}}$ and $\mathrm{VGH}_{\mathrm{ZIP}}$ based on the depiction in Figure 33. The maximal distance vector of {$t_1$, $t_2$} is $\mathrm{DV}_{\mathrm{t1,t2}}$ = [0,1,0]. This is the same as the distance vector between the two tuples.

The maximal distance vector of {$t_1$, $t_2$, $t_3$} is [0,1,2]. The maximal distance vector of {$t_1$, $t_2$, $t_4$} is [0,1,1]. The maximal distance vector of {$t_2$, $t_3$, $t_4$} is [0,1,2]. And, the maximal distance vector of {$t_1$, $t_2$, $t_3$, $t_4$} is [0,1,2].

**Theorem 7**

Given a table T[QI] and a set of tuples $S = \{t_1[\text{QI}], \ldots, t_m[\text{QI}]\}$ where $\forall i=1,\ldots,m$, $t_i \in \text{T}$, the maximal distance vector $\text{DV}_\text{S}$ across *S* is the minimal distortion required to make the tuples of *S* indistinguishable over QI.

Proof.

Let $\text{DV}_\text{S}$ be the maximal distance vector across *S*.

Assume $\text{DV}_\text{S}$ does not represent a minimal distortion of *S*.

Then there must exist a distance vector DV' that provides a minimal distortion of *S* such that **dist**(DV') < **dist**($\text{DV}_\text{S}$).

$\exists A_i \in \text{QI}$, $d_i' < d_{si}$ where $\text{DV}'=[\ldots,d_i',\ldots]$ and $\text{DV}_\text{S} = [\ldots,d_{si},\ldots]$.

This is a contradiction because $d_i' = d_{si}$.

So, $\text{DV}_\text{S}$ must be a minimal distortion of *S*.

### 8.2.2 The *k*-Similar algorithm

This subsection begins with a general description of the overall operation of the algorithm. Following this high-level description is the algorithmic listing of the *k*-Similar algorithm along with supporting algorithms. After the listings is a walk through the algorithm, without and then with examples.

The basic phases of the *k*-Similar algorithm are provided in Figure 88. The program begins in phase A by testing for some base conditions, which are: (1) if the number of tuples in the table is 0, the empty table is returned; (2) if the number of tuples in the table is less than *k*, an error results; and, (3) if the number of tuples in the table is greater than or equal to *k*, but less than 2*k*, all the tuples are generalized into one cluster that is returned as the solution.

In all other cases, the program continues by automatically computing distance vectors between every two tuples and organizing the result into a clique. Each distance vector recorded on an edge of the clique reports the generalization needed in order for the two incident tuples to have the same generalized result.

In phase B, the program walks the edges of the clique to identify groups of *k* tuples that are "closest" in terms of distance vectors. A set of *k* tuples that are minimally distant denote a possible cluster of tuples in the generalized solution. Each of tuple in the cluster appears in the generalized solution with the same generalized values. The set of all *k*-sized clusters determined to minimally include a tuple is called *mins*. Each cluster is called a "minimal". The remainder of the algorithm works with *mins* and subsets and partitions of *mins* to identify which group of clusters in *mins* best accounts for all the tuples that when generalized in accordance to their designated clusters would yield minimal distortion in the overall generalized solution.

Some of the clusters in *mins* may consist of tuples that if their attributes were generalized to the same values would not limit the ability of other tuples to combine with their closest tuples. I term such a cluster a "complementary minimum". In phase C, the program traverses through *mins* identifying any complementary minimums. Phase D handles the situation if complementary minimums are found in *mins* and phase E handles the situation if no complementary minimums are found.

In phase D, if complementary minimums exist in *mins*, then each such cluster is removed from further consideration. That is, the tuples that comprise a complementary minimum are generalized together and added to the generalized solution. Recall, a cluster in *mins*, from phase B, identified its constituent tuples as being minimally distant and the cluster as containing *k* tuples. Therefore, if the cluster is a complementary minimum, it provides a solution for its constituent tuples.

Clusters remaining in *mins,* after complementary minimums are removed, have groups of clusters that share tuples. The program is recursively run on each connected partition of the remaining clusters in *mins*.

Phase E concerns partitions of *mins* that have no complementary minimums. This is a special situation in which groups of clusters share one or more common tuples. These common tuples are held aside and the program recursively run on the result. When execution returns from the recursion, the tuples, which were previously held aside, are added to the results so that the overall distortion is minimal.

Basic operation of the *k*-Similar algorithm is as follows:

A. Compute distance vectors between every two tuples in the table T[QI]. The result is a clique and is called clique.

B. Walk the edges of clique and identify the (*k*-1) tuples that are minimally distant from each tuple. A set of tuples that are closest, based on *dist()* applied to their maximal distance vector, is termed a "minimal". The resulting set of "minimals" for all tuples is called *mins*.

C. Identify elements of *mins* that are isolated from other minimals in *mins*. Such elements represent tuples that if they are excluded from the clique would not limit other tuples from combining with their closest tuples. Such a set of tuples is termed a "complementary minimum". The set of all complementary minimums found in *mins* is called *complements*.

D. If complementary minimums exist in *mins*, then for each element of *complements*: (1) put the corresponding tuples in the solution table, all minimally generalized to be indistinguishable; and, (2) remove those tuples from further consideration. Recursively run the program on connected partitions of the tuples remaining.

E. If no complementary minimums exist, then there exist a set of 1 to (*k*-1) tuples that are common to all minimals in *mins*. In this case, remove the common tuple(s) from consideration. Recursively run the program on the result and then add the withheld tuple(s) so that the overall distortion after the withheld tuple(s) are included is minimal.

**Figure 88 Basic operation of *k*-Similar algorithm**

Figure 89 contains a listing of the *k*-Similar algorithm. Figure 90 through Figure 102 provide supporting methods. A description of the general operation of the algorithm and examples using these algorithms are provided following the algorithm listings.

```
k-Similar Algorithm
Input:      Table T; quasi-identifier QI = ($A_1$, …, $A_n$), k-anonymity constraint k; and domain and
            value generalization hierarchies $DGH_{Ai}$ and $VGH_{Ai}$, where i=1,…,n with accompanying
            functions $f_{Ai}$.
Output:     A k-minimal distortion of T[QI]
Assume:     |T| ≥ k
algorithm k-Similar:
1.  Append an attribute ID to T. The associated values of ID in T are key identifiers that are unique for
    each tuple of T; these values are numbered from 1 to |T|.
2.  clique = CliqueConstruct( T[QI,ID] )
3.  clusts ← kSimilarRun(T, k, clique)
4.  return TableConstruct(clusts)
```

**Figure 89 *k*-Similar algorithm**

```
CliqueConstruct
Input:      Table T[QI,ID]; where quasi-identifier QI = ($A_1$, …, $A_n$), ID associates unique values
            numbered from 1 to |T| to the tuples of T, and value generalization hierarchies $VGH_{Ai}$
            and $VGH_{Ai}$, where i=1,…,n with accompanying functions $f_{Ai}$.
Output:     clique, which is a clique of the tuples of T stored in a 2-dimensional array. Each node in
            the clique is a tuple. Each edge records the distance vector that corresponds to the
            distance between the tuples whose nodes are incident.
algorithm CliqueConstruct:
1.  let clique be an initially empty 2-dimensional square array of size |T| by |T|.
    2.1   for tuplefrom ← 1 to |T| do:
          2.1.1   for tupleto ← 1 to |T| do:
                  2.1.1.1   if (tuplefrom ≠ tupleto) then:
                            2.1.1.1.1   clique[tuplefrom, tupleto]
                                              ← Distance(T[QI,ID=tuplefrom], T[QI,ID=tupleto])
2.  return clique
```

**Figure 90 CliqueConstruct algorithm**

**Distance**

**Input:** $t_1,t_2 \in$ T[QI]; where quasi-identifier QI = $(A_1, \ldots, A_n)$, and value generalization hierarchies $\mathrm{VGH}_{A_i}$, where $i$=1,…,$n$ with accompanying functions $f_{Ai}$.

**Output:** $[d_1, \ldots, d_n]$, which is a distance vector that corresponds to the distance between the tuples $t_1$ and $t_2$.

**algorithm Distance:**

```
DV ← $[d_1,\ldots,d_n]$ where each $d_i$ is the length of the unique path between $t_1[A_i]$ and $t_2[A_i]$ in $\mathrm{VGH}_{A_i}$ for i=1…$n$
return DV
```

**Figure 91 Distance vector algorithm**

***k*SimilarRun Algorithm**

**Input:** Table T[QI,ID], where quasi-identifier QI = $(A_1, \ldots, A_n)$, ID associates unique values numbered from 1 to |T| to the tuples of T; *k*-anonymity constraint *k*; value generalization hierarchies $\mathrm{VGH}_{A_i}$, where $i$=1,…,$n$ with accompanying functions $f_{Ai}$; and, clique, which is a clique of the tuples of T where each node in the clique is a tuple and each edge records the distance vector that corresponds to the distance between the tuples whose nodes are incident.

**Output:** *clusts*, which is a vector of sets of ID values of tuples. Each member set identifies a cluster of tuples that when generalized to respect to the distance vectors incident to the tuples provide a set of "closest" tuples in a *k*-minimal distortion of T[QI]

**algorithm *k*SimilarRun:**

```
if |T| = 0 then return ∅
if |T| < k then error "Table must have at least k elements"
if |T| < 2*k then return { T[ID] }                // make a cluster containing all tuples in T
mins ← GenerateMinimums(T[QI,ID], clique, k)
complements ← FindComplements(mins)
if |complements| > 0 then do:
    let $T_2$ be a table with no elements initially
    for pos ← 1 to |complements| do:
        $T_2$ ← {t[QI,ID] | t[QI,ID]∈T[QI,ID∈ complements[pos]] }
        T ← T – $T_2$
        if ( |T| > 0) then do: mins ← GenerateMinimums(T[QI,ID], clique, k)
return complements ∪ kSimilarRunParts( T, mins )
```

**Figure 92 *k*SimilarRun algorithm**

***k*SimilarRunParts Algorithm**

**Input:** Table T[QI,ID]; where quasi-identifier QI = ($A_1$, …, $A_n$), ID associates unique values numbered from 1 to |T| to the tuples of T, and *mins*, which is a vector of sets of ID values of tuples. Each member set identifies a cluster of *k* closest tuples.

**Output:** *clusts*, which is a vector of sets of ID values of tuples. Each member set identifies a cluster of tuples that when generalized to respect to the distance vectors incident to the tuples provide a set of "closest" tuples in a *k*-minimal distortion of T[QI]. Executes *kSimilarRun()* mutually recursively, on connected groups within *mins*.

**algorithm *k*SimilarRunParts:**

```
if (T ≡ ∅) then return ∅
($T_1$, mins1, $T_2$, mins2) ← Partition(T, mins)
if ( $|T_1| < 2*k$ ) then do:
        return kSimilarRun($T_1$) ∪ kSimilarRunParts($T_2$, mins2)
else do:
    // assert: there exist tuple(s) common to all elements within partition $T_1$, based on mins1
        withheld ← CommonTuples(mins1, clique)
        if ( ($|T_1|$ - |withheld|) < 2*k) then do:
            return addTuple(withheld, k,(mins1-withheld), clique)
                                ∪ kSimilarRunParts($T_2$, mins2)
        mins3 ← kSimilarRun($T_1$[QI,ID∉ withheld], k, clique)
        return addTuple(withheld, k, mins3, clique) ∪ kSimilarRunParts($T_2$, mins2)
```

**Figure 93 *k*SimilarRunParts algorithm**

**TableConstruct**

**Input:** *clusts*, which is a vector of sets of ID values of tuples. Each member set identifies a cluster of tuples that when generalized to respect to the distance vectors incident to the tuples provide a set of "closest" tuples in a *k*-minimal distortion of T[QI], where quasi-identifier QI = ($A_1$, …, $A_n$), ID associates unique values numbered from 1 to |T| to the tuples of T, and clique, which is a clique of the tuples of T where each node in the clique is a tuple and each edge records the distance vector that corresponds to the distance between the tuples whose nodes are incident.

**Output:** GT, which is a minimal generalization of T[QI]. Tuples identified within an element of *clusts* are generalized to have the same values.

**algorithm TableConstruct:**

```
1.  let GT ← ∅
2.  for clustnum ← 1 to |clusts| do:
    2.1  let V           be a distance vector of the form [$d_1$, …,$d_n$] where each $d_i$=0
                         and n is the number of attributes in the quasi-identifier QI = ($A_1$, …, $A_n$)
    2.2  let aclust      be an expandable and collapsible Vector whose elements
                         are initialized to clusts[clustnum]
    2.3  for tupleto ← 2 to |aclust| do:
         2.4.1  V ← V ⊕ clique[ aclust[1], aclust[tupleto] ]          // compute maximal distance vector
    2.4  for t ← 1 to |aclust| do:
         2.5.1  GT ← GT ∪ GeneralizeTuple(T[QI,ID=t], V)     // generalize each tuple in cluster
3.  return GT
```

**Figure 94 TableConstruct algorithm**

**AddTuple**

**Input:** *withheld*, which is a set of unique values associated with tuples in T; *k*-anonymity constraint *k*; *clusts*, also known as *mins*, is a vector of sets of ID values of tuples. Each member set identifies a cluster of tuples that when generalized to respect to the distance vectors incident to the tuples provide a set of "closest" tuples in a *k*-minimal distortion of T[QI], where quasi-identifier QI = ($A_1$, …, $A_n$), ID associates unique values numbered from 1 to |T| to the tuples of T; and, clique, which is a clique of the tuples of T where each node in the clique is a tuple and each edge records the distance vector that corresponds to the distance between the tuples whose nodes are incident.

**Output:** *clusts*, which is a vector of sets of ID values of tuples that is the same as the original value of *clusts* (also known as *mins*) provided to the algorithm except the returned value has an element that includes the elements of *withheld*. The tuple(s) identified in *withheld* replace tuple(s) in an original element of *clusts* such the overall loss of precision due to generalization is minimized and all tuples remain included.

**algorithm AddTuple:**

```
1. let $d \leftarrow \infty$, $n \leftarrow 0$, $c \leftarrow \emptyset$
2. for $clustnum \leftarrow 1$ to $|clusts|$ do:
    1.1 if $clusts[clustnum] \equiv 2 * k - |withheld|$ then do:
        1.1.1 $testclust \leftarrow$ be an expandable and collapsible Vector whose elements
                                    are initialized to $clusts[clustnum]$
        1.1.2 $(d_1, c_1) \leftarrow$addTupleMin($withheld, testclust, k, d, c,$ clique)
        1.1.3 if $(d_1 < d)$ then do:
            1.1.1.1. $d \leftarrow d_1$
            1.1.1.2. $n \leftarrow clustnum$
            1.1.1.3. $c \leftarrow c_1$
3. $temp \leftarrow clusts[n] \cup withheld$
4. $clusts[n] \leftarrow temp - c$
5. $clusts[\ |clusts|+1\ ] \leftarrow c$
6. return $clusts$
```

**Figure 95 AddTuple algorithm**

**AddTupleMin**

**Input:** $c_a$, $c_b$, which are each a set of unique values associated with tuples in T;
$k$, which is a $k$-anonymity constraint;
$d$, which is distance;
$c$, which is a set of unique values associated with tuples in T.
clique, which is a clique of the tuples of T where each node in the clique is a tuple and each edge records the distance vector that corresponds to the distance between the tuples whose nodes are incident.

**Output:** ($d$, $c$), which is a vector of sets of ID values of tuples that is the same as the original value of *clusts* (also known as *mins*) provided to the algorithm except the returned value has an element that includes the elements of *withheld*. The tuple(s) identified in *withheld* replace tuple(s) in an original element of *clusts* such the overall loss of precision due to generalization is minimized and all tuples remain included.

**Assumes** **dist()** function exists and computes non-negative distance from a distance vector based on *Prec*(), can be weighted or not.

```
algorithm AddTupleMin:
1. if $|c_a| \equiv k$ then do:
    1.1. let $V_a$, $V_b$   be distance vectors of the form $[d_1, \ldots, d_n]$ where each $d_i$=0
                           and $n$ is the number of attributes in the quasi-identifier QI = $(A_1, \ldots, A_n)$
    1.2. for $tnum \leftarrow 2$ to $|c_a|$ do:
        1.2.1 $V_a \leftarrow V_a \oplus$ clique[ $c_a[1]$, $c_a[tnum]$ ]
    1.3. $d_a \leftarrow$ dist($V_a$) * $|c_a|$
    1.4. for $tnum \leftarrow 2$ to $|c_b|$ do:
        1.4.1 $V_b \leftarrow V_b \oplus$ clique[ $c_b[1]$, $c_b[tnum]$ ]
    1.5. $d_b \leftarrow$ dist($V_b$) * $|c_b|$
    1.6. if $(d_a + d_b) < d$) then do:   return $(d_a + d_b, c_a)$
    1.7. else return ($d$, c)
2. else if $|c_a| < (k-1)$ then do:
    2.1. let $c_{a2} \leftarrow c_a$, $c_{b2} \leftarrow c_b$
    2.2. $c_{a2}$[ $|c_{a2}|$+1 ]  = $c_{b2}[1]$
    2.3. purge $c_{b2}[1]$
    2.4. $(d_1, c_1) \leftarrow$ addTupleMin($c_{a2}, c_{b2}, k, d, c$)
    2.5. if ($d_1 < d$) then do:   $d \leftarrow d_1$,   $c \leftarrow c_1$
    2.6. $(d_1, c_1) \leftarrow$ addTupleMin($c_{a2}, c_{b2}, k, d, c$)
    2.7. if ($d_1 < d$) then do:   $d \leftarrow d_1$,   $c \leftarrow c_1$
    2.8. return ($d$, $c$)
3. else while $|c_b| > 0$ do:
    3.1. $c_a$[ $|c_a|$+1 ]$\leftarrow c_b[1]$
    3.2. purge $c_b[1]$                // $c_b$ has one less element
    3.3. $(d_1, c_1) \leftarrow$ addTupleMin($c_a, c_b, k, d, c$)
    3.4. purge $c_a$[ $|c_a|$ ]              // $c_a$ has one less element
    3.5. if ($d_1 < d$) then do:   $d \leftarrow d_1$,   $c \leftarrow c_1$
4. return ($d$, $c$)
```

**Figure 96 addTupleMin algorithm**

**GeneralizeTuple Algorithm**

**Input:** Tuple $t$[QI,ID]; where quasi-identifier QI = ($A_1$, …, $A_n$), ID associates unique values numbered from 1 to |T| to the tuples of T, a distance vector V[$d_1$,…,$d_n$], and value generalization hierarchies $VGH_{Ai}$, where $i$=1,…,$n$ with accompanying functions $f_{Ai}$.

**Output:** $G$, which is a set containing the result of generalizing tuple $t$ by V.

**algorithm GeneralizeTuple:**

```
let G ← { $t_2$[QI] | $t_2[A_i] = f_1(…f_{di}(v)…)$ where $v = t[A_i]$ and V[…$d_i$…] for all $i$=1,…,|QI| }
return G
```

**Figure 97 GeneralizeTuple algorithm**

**GenerateMinimums Algorithm**

**Input:** Table T[QI,ID]; where quasi-identifier QI = ($A_1$, …, $A_n$), ID associates unique values numbered from 1 to |T| to the tuples of T, $k$-anonymity constraint $k$, and clique, which is a clique of the tuples of T where each node in the clique is a tuple and each edge records the distance vector that corresponds to the distance between the tuples whose nodes are incident.

**Output:** *mins*, which is a Vector of sets of ID values of tuples. Each member set identifies a cluster of $k$-*1* of $t$'s closest tuples. Each member set includes $t$ so the total cluster size is $k$.

**algorithm GenerateMinimums:**

```
let mins be an expandable and collapsible Vector with no elements initially.
let stack be an empty Stack.
let zero be a distance vector [$d_1$, …,$d_n$] where each $d_i$=0 and $n$ is the number of attributes in the
quasi-identifier QI = ($A_1$, …, $A_n$)
for tupleto ← 1 to |clique| do:
    mins = traverse(tupleto, tupleto+1, k, {tupleto}, zero, ∞, mins)
                    // stack and clique are globally available across iterations of traverse()
return mins
```

**Figure 98 GenerateMinimums algorithm**

**FindComplements Algorithm**

**Input:** *mins*, which is a set of sets of ID values of tuples. Each member set identifies a cluster of *k-1* of *t*'s closest tuples. Each member set includes *t* so the total cluster size is *k*.

**Output:** *distincts*, which is a vector of sets of ID values of tuples. Each member set identifies a cluster that can be partitioned as an independent sub-solution.

**algorithm FindComplements:**

```
let distincts be an expandable and collapsible Vector with no elements initially.
let $allnodes \leftarrow \emptyset$
for $pos \leftarrow 1$ to $|mins|$ do:
    $allnodes \leftarrow allnodes \cup mins[pos]$
for $candidate \leftarrow 1$ to $|mins|$ do:
    let $s \leftarrow allnodes - mins[candidate]$
    for $pos \leftarrow 1$ to $|mins|$ do:
        $temp \leftarrow mins[pos] \cap mins[candidate]$
        if ( $temp \neq \emptyset$) then do:
            $s \leftarrow s - temp$
    if ($s \equiv allnodes - mins[candidate]$ ) then do:
        $distincts[\ |distincts| + 1] \leftarrow mins[candidate]$
return distincts
```

**Figure 99 FindComplements algorithm**

**Traverse Algorithm**

**Input: (*node, next, k, path, mV, mdist, mins*)**

*node* which is the unique value associated with a tuple in clique that represents the tuple "from" which distance will be measured to *next* on this iteration.

*next* which is the unique value associated with a tuple in clique that represent the tuple "to" which distance will be measured from *node* on this iteration.

*k* which is the *k*-anonymity constraint

*path* which is the set of tuples comprising the shortest path from *node* to the tuple that serves as the root of the traversal

*mV* which is a maximal distance vector from the tuple that serves as the root of the traversal to *node.*

*mdist* which is the measure of distortion from the root of the traversal to *node*. It does not include the distance from *node* to *next.*

*mins* which is a Vector of sets of ID values of tuples computed so far. Each member set identifies a cluster of *k-1* of *t*'s closest tuples. Each member set includes *t* so the total cluster size is *k*. At the end of the traversal this value provides the answer. It is shared across iterations to track global information.

**Output:** *mins,* which is a Vector of sets of ID values of tuples. Each member set identifies a cluster of *k-1* of *t*'s closest tuples. Each member set includes *t* so the total cluster size is *k*.

**Assumes** **dist()** function exists and computes non-negative distance from a distance vector based on *Prec*(), can be weighted or not.

Assumes following exist and are globally available:

*stack* which is a Stack that contains information on each node from the root of the traversal up to, but not including *node*. Each element of the stack contains values of the form: (*node, path, mV, mdist*). It is shared across iterations to track global information.

clique which is a clique of the tuples of T where each node in the clique is a tuple and each edge records the distance vector that corresponds to the distance between the tuples whose nodes are incident, *t*, which is an ID value unique to a tuple in T.

```
algorithm Traverse:
1. if ($next$ > |clique|) and stackEmpty() then do:
        1.1.  return $mins$
2. else if ($next$ > |clique|) then do:
        2.1.  $(root_0, path_0, mV_0, mdist_0) \leftarrow$ stackPop()
        2.2.  return traverse($root_0$, $node+1$, $k+1$, $path_0$, $mins$)
3. else if ($next \notin$ T[ID] ) then do:
        3.1.  return traverse($node$, $next+1$, 1, $path$, $mV$, $mdist$, $mins$)
4. $V_1 \leftarrow mV \oplus$ clique[$node$, $next$]
5. $d_1 \leftarrow$ dist($V_1$) * (|$path$| + 1)
6. $p_1 \leftarrow path \cup \{next\}$
7. if ($d_1 > mdist$) then do:
        7.1.  return traverse($node$, $next+1$, $k$, $path$, $mV$, $mdist$, $mins$)
8. else if ($k \equiv 1$) and ($d_1 \equiv mdist$) then do:
        8.1.  $mins$[ |$mins$| + 1] $\leftarrow p_1$
        8.2.  return traverse($node$, $next+1$, 1, $path$, $mV$, $mdist$, $mins$)
9. else if ($k \equiv 1$) then do:          // and ($d_1 < mdist$) is implied
        9.1.  purge all elements from $mins$
        9.2.  $mins[1] \leftarrow p_1$
        9.3.  $mdist \leftarrow d_1$
        9.4.  $mV \leftarrow V_1$
        9.5.  return traverse($node$, $next+1$, 1, $path$, $mV$, $mdist$, $mins$)
10. else do:                                // $k \neq 1$ is implied
        10.1. stackPush($next$, $p_1$, $V_1$, $d_1$)
        10.2. return traverse($next$, $next+1$, $k-1$, $p_1$, $V_1$, $d_1$, $mins$)
```

**Figure 100 Traverse algorithm**

**Partition Algorithm**

**Input:** Table T[QI,ID]; where quasi-identifier QI = ($A_1$, …, $A_n$), ID associates unique values numbered from 1 to |T| to the tuples of T; and, *mins*, which is a set of sets of ID values of tuples. Each member set identifies a cluster of *k-1* of *t*'s closest tuples. Each member set includes *t* so the total cluster size is *k*.

**Output:** ($T_1$, $T_2$, *ms*), where $T_1 \cup T_2 = T$ and $T_1 \cap T_2 = \emptyset$. The tuples of $T_1$ identifies a connected group of tuples that can be partitioned as an independent sub-solution. This decision is based on the connectedness of elements within *mins*. The identifier *ms* contains the subset of *mins* not accounted for by the tuples of $T_1$.

**algorithm Partition:**

```
1.  let $allnodes \leftarrow \emptyset$, $ms \leftarrow \emptyset$
2.  for $pos \leftarrow 1$ to $|mins|$ do:
    2.1   $allnodes \leftarrow allnodes \cup mins[pos]$
3.  let $r \leftarrow mins[1]$                // test connectedness of $mins[1]$
4.  for $pos \leftarrow 2$ to $|mins|$ do:
    4.1   if ( $mins[pos] \cap r \neq \emptyset$) then do:
          4.1.1   $r \leftarrow r \cup mins[pos]$
    4.2   else do:
          4.2.1   $ms \leftarrow ms \cup mins[pos]$
5.  if ( $mins \neq r$ ) then do:
    5.1   return ($T_1$, $r$, $T_2$, $ms$) where $T_1 = \{t_1 \mid t_1 \in T[QI,ID=t_2]$ and $t_2 \in r\}$ and $T_2 = T - T_1$
6   else do:
    6.1   return (T, $r$, $\emptyset$, $\emptyset$)
```

**Figure 101 Partition algorithm**

**CommonTuples Algorithm**

**Input:** *mins*, which is a set of sets of ID values of tuples. Each member set identifies a cluster of *k-1* of *t*'s closest tuples. Each member set includes *t* so the total cluster size is *k*; and, clique, which is a clique of the tuples of T where each node in the clique is a tuple and each edge records the distance vector that corresponds to the distance between the tuples whose nodes are incident, *t*, which is an ID value unique to a tuple in T

**Output:** *withheld*, which is a set of unique value associated with a tuple in T and that occurs in each element of *mins* thereby making them "the" closest tuple to all tuples.

**algorithm CommonTuples:**

```
1.  let withheld ← ∅
2.  for tnum ← 1 to |clique| do:
    2.1  let inall ← true
    2.2  for pos ← 1 to |mins| do:
         2.2.1  if (tnum ∉ mins[pos])
                2.2.1.1  inall ← false
    2.3  if ( inall ≡ true ) then do:
         2.3.1  withheld ← withheld ∪ {tnum}
3.  return withheld
```

**Figure 102 CommonTuples algorithm**

As introduced earlier, the basic steps, A through E, of the *k*-Similar algorithm are enumerated in Figure 88. The algorithm listed in Figure 89 along with its supporting methods is more detailed but follows these same basic steps. Below is a walk through the detailed version of the *k*-Similar algorithm. Afterwards are some examples.

Given a private table T, a quasi-identifier QI=($A_1$,…,$A_n$), a *k*-anonymity requirement *k*, domain and value generalization hierarchies $\text{DGH}_{\text{Ai}}$ and $\text{VGH}_{\text{Ai}}$, where *i*=1,…,*n* with accompanying functions $f_{Ai}$, the *k*-Similar algorithm, listed in Figure 89, generates a *k*-minimal distortion of T[QI].

The *k*-Similar algorithm listed in Figure 89 begins in step 1 by expanding T to include an attribute labeled *ID* whose values serve as a unique identifier (or key) for each tuple in T. From this point forward, the algorithm has the ability to uniquely refer to a tuple in T by using its associated value of *ID*.

Step 2 of the *k*-Similar algorithm listed in Figure 89 produces a clique of the tuples of T stored in a 2-dimensional array named clique. The method *CliqueConstruct()* listed in Figure 90 performs the construction. Each node in the clique is a tuple. Each edge records the distance vector that corresponds to the distance between the tuples whose nodes are incident. The method *Distance()* listed in Figure 91 computes the distance vector between two tuples using the value generalization hierarchies $VGH_{Ai}$, where $i$=1,…,$n$ with accompanying functions $f_{Ai}$. The distance vector records the minimal generalization strategy [122] needed for the two tuples to have the same generalized values.

The heart of the algorithm occurs in step 3 of the *k*-Similar algorithm listed in Figure 89. It executes the method *kSimilarRun()*, which is listed in Figure 92, and which will be further described in the next paragraphs. The *kSimilarRun()* method returns a set of clusters of tuples such that minimally generalizing the tuples of each cluster together so they become indistinguishable results in a table that is a *k*-minimal distortion of T[QI]. The method *TableConstruct()* listed in Figure 94 takes the set of clusters from *kSimilarRun()*, generalizes the tuples of each cluster, and then returns the generalized table. Each cluster therefore, identifies a group of tuples that in the solution set are indistinguishable across QI. So, the *k*-Similar approach can be described as translating the problem into one of partitioning tuples. This is done by *kSimilarRun()*.

The *kSimilarRun()* method listed in Figure 92 begins by testing for the base conditions in steps 1 through 3. These conditions are based on the size of the table provided to *kSimilarRun()*. Step 1: if the number of tuples in the table is 0, an empty set of clusters is returned denoting the empty table. Step 2: if the number of tuples in the table is less than *k*, an error results because the *k* requirement cannot be satisfied on a table having less than *k* tuples. Step 3: if the number of tuples in the table is greater than or equal to *k*, but less than 2*k*, all the tuples are generalized into one cluster designating that all the tuples of the table are to be generalized together.

In step 4 of the *kSimilarRun()* method, which is listed in Figure 92, the program walks the edges of clique using the method *GenerateMinimums()*, which is listed in Figure 98, to identify groups of *k* tuples that are "closest" in terms of distance vectors. The method *traverse()*, which is listed in Figure 100, performs the actual traversal on clique given a particular starting tuple *t*. The method *traverse()* returns the cluster(s) of size *k* containing *t* and *t*'s closest tuples that when combined have less distortion than any other combination of *k* tuples that include *t*. The method *GenerateMinimums()* executes *traverse()* on each tuple. The end result is a set of all *k*-sized clusters determined to minimally include a tuple. It is called *mins*. Each cluster in *mins* is called a "minimal". As described in the next paragraphs, the remainder of the algorithm works with *mins* and partitions of *mins* to identify which group of clusters in *mins* best accounts for all the tuples that when generalized in accordance to their designated clusters would yield minimal distortion in the overall generalized solution.

Some of the clusters in *mins* may consist of tuples that if their attributes were generalized to the same values would not limit the ability of other tuples to combine with their closest tuples. I term such a cluster a "complementary minimum". Step 5 of the *kSimilarRun()* method, which is listed in Figure 92, executes the *FindComplements()* method, which is listed in Figure 99, to identify complementary minimums within *mins*. Such clusters can be partitioned as an independent sub-solution. The resulting set of complementary minimums found is called *complements*.

The sub-steps of step 6 of the *kSimilarRun()* method, which is listed in Figure 92, execute only if complementary minimums are found in *mins*. In that case, *complements* returns as part of the solution and *kSimilarRunParts(),* which is listed in Figure 93, executes on the remaining tuples and minimals to recursively apply the algorithm on partitions of connected clusters. If no complementary minimums are found, then *complements* has no elements, and so in step 7, *kSimilarRunParts()*, which is listed in Figure 93, executes on all the tuples and minimals under consideration.

The method *kSimilarRunParts(),* which is listed in Figure 93, employs mutual recursion by executing *kSimilarRun()* on each connected partition of the remaining clusters in *mins*. The method *Partition(),* which is listed in Figure 101, is used in step 2 of *kSimilarRunParts()* to identify connected clusters within the given *mins*. If the returned partition has less than 2*k* elements, then in step 3.1, *kSimilarRun()* is used to combine the tuples of that partition into a single cluster as part of the overall solution.

If the returned partition, identified as $T_1$, has 2*k* or more elements, then the partition has a special configuration in which all minimals within the partition share one or more common tuples. This situation is handled in step 4 of *kSimilarRunParts()*. In step 4.1, the method *kSimilarRunParts()* deploys the method *CommonTuples(),* which is listed in Figure 102, to identify the set of 1 to (*k*-1) tuples that appear within each cluster of the partition. These tuples are stored in a set called *withheld*. If the number of tuples in the partition, not including the tuples withheld, is less than 2*k*, then the method *addTuple(),* which is listed in Figure 95, executes to determine which clusters in the partition should include the withheld tuples. The decision is made so that the overall result has minimal distortion. On the other hand, if the number of tuples in the partition, not including the tuples withheld, is greater than or equal to 2*k*, then *kSimilarRun()* is executed using mutual recursion on the partition not including the withheld tuples.

The method *addTuple()* then executes afterwards to determine which cluster(s) in the result will include the withheld tuples.

As stated earlier, the final step of the *k*-Similar algorithm uses *TableConstruct()*, which is listed in Figure 94, to construct a generalized table from the resulting set of clusters from *kSimilarRun()*. It can be shown that the final table resulting from the *k*-Similar algorithm is a *k*-minimal distortion of the original table using cell-level generalization and suppression.

**Example**

Given the private table PT shown in Figure 84, the domain and value generalization hierarchies based on the depictions in Figure 33 (on page 101), and a *k*-anonymity requirement of *k*=2, the *k*-Similar algorithm, which is listed in Figure 89, provides the table GT, as shown in Figure 104, as a *k*-minimal distortion of PT over the quasi-identifier QI = {*HomeZIP*, *HospitalZIP*, *WorkZIP*}. Here is a walk through the *k*-Similar algorithm to demonstrate how MT is constructed.

Figure 84 shows the uniquely identifying values *t1*, *t2*, *t3* and *t4* appended to the table after step 1 of the *k*-Similar algorithm executes. These values are associated with the *ID* attribute. Figure 85 shows clique, which is constructed after step 2 of the *k*-Similar algorithm concludes. The nodes are the tuples of PT. The edges are labeled with the distance vectors between every two tuples in PT.

None of the base conditions in the first 3 steps of *kSimilarRun()* are applicable. T in this case is PT. It has 4 tuples and *k*=2, so |T|=2*k*. Figure 103 shows the value of *mins* after step 4 concludes. The method *GenerateMinimums()* identifies the set of minimals for each tuple by traversing clique to identify each tuple's nearest (*k*-1) tuples. Traversing clique from *t1* provides the minimal {*t1*, *t2*}, from *t2* provides the minimals {*t1*, *t2*} and {*t2*, *t4*}, from *t3* provides the minimal {*t1*, *t3*}, and from *t4* provides the minimal {*t2*, *t4*}.

| |
|---|
| {*t1*, *t2*} |
| {*t2*, *t4*} |
| {*t1*, *t3*} |

**Figure 103 Resulting *mins* from *GenerateMinimums()***

The minimals {*t1*, *t3*} and {*t2*, *t4*} are returned as complementary minimums by *FindComplemets()*. So, *complements* = {{*t1*, *t3*}, {*t2*, *t4*}} after step 5 of *kSimilarRun()*. When step 6 of *kSimilarRun()* concludes, T is empty. So, *complements* is returned at step 7 of *kSimilarRun()* as the set of clusters that are minimally distorting. The call to *kSimilarRunParts()* in step 7 of *kSimilarRun()* returns ∅ because T is empty. The final step of *kSimilar()* executes *TableConstruct()* on *clusts* ={{*t1*, *t3*}, {*t2*, *t4*}}. The result is shown in Figure 104 with the ID values still appended for ease of reference.

The possible cluster combinations and their distortion are: {{*t1*, *t2*}, {*t3*, *t4*}} at 8 levels of generalization is 2.67; {{*t1*, *t3*}, {*t2*, *t4*}} at 6 levels of generalization is 2.00; and, {{*t1*, *t4*}, {*t2*, *t3*}} at 10 levels of generalization is 3.33. The combination of clusters with the least distortion is {{*t1*, *t3*}, {*t2*, *t4*}}, which is the same found by *kSimilar()*.

| | *A1* | *A2* | *A3* |
|---|---|---|---|
| | **Home ZIP** | **Hospital ZIP** | **Work ZIP** |
| *t1* | 02138 | 02138 | 021** |
| *t2* | 02138 | 02139 | 0213* |
| *t3* | 02138 | 02138 | 021** |
| *t4* | 02138 | 02139 | 0213* |

**Figure 104 Result from *k*-Similar applied to PT in Figure 84**

### Example

Given the private table PT shown in Figure 34 (on page 102), the domain and value generalization hierarchies based on the depictions in Figure 33 (on page 101), a *k*-anonymity requirement of *k*=2, the *k*-Similar algorithm, which is listed in Figure 89, provides the table GT, as shown in Figure 104, as a *k*-minimal distortion of PT over the quasi-identifier QI = {*Race*, *BirthDate*, *Gender*, *ZIP*}. Here is a walk through the *k*-Similar algorithm to demonstrate how MT is constructed.

Figure 34 shows the uniquely identifying values *t1*, *t2*, *t3*, …, *t12* appended to the table after step 1 of the *k*-Similar algorithm executes. These values are associated with the *ID* attribute. Figure 105 shows clique, which is constructed after step 2 of the *k*-Similar algorithm concludes. The nodes are the tuples of PT. The edges are labeled with the distance vectors between every two tuples in PT. The clique is stored in Figure 105 as a 2-dimensional array. Each row and each column represent a tuple. The cell located at row $t_i$ and column $t_j$ stores the distance vector $V_{t1,t2}$.

| | *t1* | *t2* | *t3* | *t4* | *t5* | *t6* | *t7* | *t8* | *t9* | *t10* | *t11* | *t12* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***t1*** | 0 0 0 0 | 0 2 0 0 | 0 2 1 2 | 0 2 1 2 | 0 4 1 2 | 0 4 1 2 | 1 4 0 2 | 1 2 1 2 | 1 4 0 2 | 1 4 0 2 | 1 3 0 2 | 1 3 0 2 |
| ***t2*** | 0 2 0 0 | 0 0 0 0 | 0 2 1 2 | 0 2 1 2 | 0 4 1 2 | 0 4 1 2 | 1 4 0 2 | 1 2 1 2 | 1 4 0 2 | 1 4 0 2 | 1 3 0 2 | 1 3 0 2 |
| ***t3*** | 0 2 1 2 | 0 2 1 2 | 0 0 0 0 | 0 2 0 0 | 0 4 0 0 | 0 4 0 0 | 1 4 1 0 | 1 2 0 1 | 1 4 1 1 | 1 4 1 1 | 1 3 1 0 | 1 3 1 0 |
| ***t4*** | 0 2 1 2 | 0 2 1 2 | 0 2 0 0 | 0 0 0 0 | 0 4 0 0 | 0 4 0 0 | 1 4 1 0 | 1 2 0 1 | 1 4 1 1 | 1 4 1 1 | 1 3 1 0 | 1 3 1 0 |
| ***t5*** | 0 4 1 2 | 0 4 1 2 | 0 4 0 0 | 0 4 0 0 | 0 0 0 0 | 0 2 0 0 | 1 2 1 0 | 1 4 0 1 | 1 2 1 1 | 1 2 1 1 | 1 4 1 0 | 1 4 1 0 |
| ***t6*** | 0 4 1 2 | 0 4 1 2 | 0 4 0 0 | 0 4 0 0 | 0 2 0 0 | 0 0 0 0 | 1 2 1 0 | 1 4 0 1 | 1 2 1 1 | 1 2 1 1 | 1 4 1 0 | 1 4 1 0 |
| ***t7*** | 1 4 0 2 | 1 4 0 2 | 1 4 1 0 | 1 4 1 0 | 1 2 1 0 | 1 2 1 0 | 0 0 0 0 | 0 4 1 1 | 0 2 0 1 | 0 2 0 1 | 0 4 0 0 | 0 4 0 0 |
| ***t8*** | 1 2 1 2 | 1 2 1 2 | 1 2 0 1 | 1 2 0 1 | 1 4 0 1 | 1 4 0 1 | 0 4 1 1 | 0 0 0 0 | 0 4 1 0 | 0 4 1 0 | 0 3 1 1 | 0 3 1 1 |
| ***t9*** | 1 4 0 2 | 1 4 0 2 | 1 4 1 1 | 1 4 1 1 | 1 2 1 1 | 1 2 1 1 | 0 2 0 1 | 0 4 1 0 | 0 0 0 0 | 0 2 0 0 | 0 4 0 1 | 0 4 0 1 |
| ***t10*** | 1 4 0 2 | 1 4 0 2 | 1 4 1 1 | 1 4 1 1 | 1 2 1 1 | 1 2 1 1 | 0 2 0 1 | 0 4 1 0 | 0 2 0 0 | 0 0 0 0 | 0 4 0 1 | 0 4 0 1 |
| ***t11*** | 1 3 0 2 | 1 3 0 2 | 1 3 1 0 | 1 3 1 0 | 1 4 1 0 | 1 4 1 0 | 0 4 0 0 | 0 3 1 1 | 0 4 0 1 | 0 4 0 1 | 0 0 0 0 | 0 2 0 0 |
| ***t12*** | 1 3 0 2 | 1 3 0 2 | 1 3 1 0 | 1 3 1 0 | 1 4 1 0 | 1 4 1 0 | 0 4 0 0 | 0 3 1 1 | 0 4 0 1 | 0 4 0 1 | 0 2 0 0 | 0 0 0 0 |

**Figure 105 Clique showing distance vectors between tuples of Figure 34**

None of the base conditions in the first 3 steps of *kSimilarRun()* are applicable. T in this case is PT. It has 12 tuples and *k*=2, so |T|>2*k*. Figure 106 shows the value of *mins* after step 4 concludes. The method *GenerateMinimums()* identifies the set of minimals for each tuple by traversing clique to identify each tuple's nearest (*k*-1) tuples. Traversing clique from *t1* provides the minimal {*t1*, *t2*}, from *t2* provides {*t1*, *t2*}, from *t3* provides {*t3*, *t4*}, from *t4* provides {*t3*, *t4*, from *t5* provides {*t5*, *t6*}, from *t6* provides {*t5*, *t6*}, from *t7* provides {*t7*, *t9*} and {*t7*, *t10*}, from *t8* provides {*t3*, *t8*} and {*t4*, *t8*}, from *t9* provides {*t9*, *t10*}, from *t10* provides {*t9*, *t10*}, from *t11* provides {*t11*, *t12*}, and from *t12* provides {*t11*, *t12*}.

| |
|---|
| {*t1*, *t2*} |
| {*t3*, *t4*} |
| {*t5*, *t6*} |
| {*t7*, *t9*} |
| {*t7*, *t10*} |
| {*t3*, *t8*} |
| {*t4*, *t8*} |
| {*t9*, *t10*} |
| {*t11*, *t12*} |

**Figure 106 Resulting *mins* from *GenerateMinimums()***

The minimals {*t1*, *t2*}, {*t5*, *t6*} and {*t11*, *t12*} are returned as complementary minimums by *FindComplemets()*. So, *complements* = {{*t1*, *t2*}, {*t5*, *t6*}, {*t11*, *t12*}} after step 5 of *kSimilarRun()*. When step 6 of *kSimilarRun()* concludes,T[ID] is {*t3*, *t4*, *t7*, *t8*, *t9*, *t10*} with *mins* = {{*t3*, *t4*}, {*t3*, *t8*}, {*t4*, *t8*}{*t7*, *t9*}, {*t7*, *t10*}, {*t9*, *t10*}}. So, *complements* is returned at step 7 of *kSimilarRun()* as a set of clusters that are minimally distorting that comprise part of the overall solution

The call to *kSimilarRunParts()* executes in step 7 of *kSimilarRun()* on the remaining tuples and minimals. The Partition() method returns the partition {{*t3*, *t4*}, {*t3*, *t8*}, {*t4*, *t8*}} and the subsequently{{*t7*, *t9*}, {*t7*, *t10*}, {*t9*, *t10*}}. Each of these are clustered together in *kSimilarRun()* to be: {*t3*, *t4*, *t8*} and {*t7*, *t9*, *t10*} because each of these partitions have less than 2*k* (or 4) tuples. The final step of *kSimilar()* executes *TableConstruct()* on *clusts* ={{*t1*, *t2*}, {*t5*, *t6*}, {*t11*, *t12*}, {*t3*, *t4*, *t8*}, {*t7*, *t9*, *t10*}}. The result is shown as GT1 in Figure 35 (see page 102). The appended *ID* values have been discarded.

In the example on page 122, MinGen(PT)=GT1. The same solution derived by *k-Similar()* was determined to be a *k*-minimal distortion of PT over QI with respect to $\text{DGH}_{\text{Ai}}$ where *Prec*(GT1)=0.83.

## 8.3 Comparison to Mingen

A comparison to MinGen [123] requires examining: (1) the computational complexity of the algorithm to ensure it operates in reasonable time; and, (2) the correctness of the algorithm. These are discussed in the following subsections.

### 8.3.1 Complexity of the *k*-Similar algorithm

The *k*-Similar algorithm listed in Figure 89 with supporting methods in Figure 90 through Figure 102 was not written as efficiently as possible. Nevertheless, here is a walk through the algorithm noting the methods that characterize the computational complexity.

The *CliqueSetup()* method, which is listed in Figure 90, is executed in step 2. Comparing every two tuples and determining their distance vector is done in $O(|T|^2)$ time. The *GenerateMinimums()* method, which is listed in Figure 98, working along with its accompanying *traverse()* method, which is listed in Figure 100, pose a serious problem for the computational speed of *k-Similar()*. As implemented, they operate in combinatoric time because every combination of |T| tuples drawn *k* at a time are examined. While some efficiencies may be possible in future versions, the version provided here is combinatoric. Clearly, this overwhelms computational complexity of the remainder of the algorithm. The efficiencies gained by partitioning the clusters into sub-clusters make the algorithm useful in some real-world applications and dramatically improves the performance over MinGen. However, combiantoric operation is not practical for most uses.

More importantly however, the techniques presented in this algorithm concerning the use and operations on distance vectors and partitions of clusters holds promise as ways to reduce the computational complexity.

### 8.3.2 Correctness of the *k*-similar algorithm

The correctness of the *k*-Similar algorithm relies on its ability to produce solutions that adhere to a given *k*-anonymity requirement, assuming of course a proper quasi-identifier and a proper value for *k* have been provided. In this subsection, I will show that the *k*-Similar algorithm provides solutions that do adhere to a given *k*-anonymity requirement. Here is a walk through the program, noting its correctness with respect to the *k* requirement.

A result from *k*-Similar properly adheres to the *k* requirement if each and every cluster provided by *kSimilarRun()* is of size *k* or more because *TableConstruct()* merely generalizes the tuples identified in each cluster provided from *kSimilarRun()*. A table must have at least *k* tuples to adhere to *k*-anonymity. So, in step 3 of *kSimilarRun()*, a table that has *k* or more tuples, but less than 2*k* tuples, results in a single cluster. This cluster is therefore of size *k* or more.

Execution of *k-SimilarRun()* continues for tables that have more than 2*k* tuples. The set of minimals produced by *GenerateMinimums()* at step 4 of *kSimilarRun()* have clusters of size *k* because *GenerateMinimums()* traverses paths of *k*-1 in clique from a given tuple and returns the path with the maximal distance vector that has the minimal distance. All minimals identified by *GenerateMinimums()* therefore has *k* elements. of the minimals returned from *GenerateMinimums()*, some are identified as complementary minimums and appended to the solution set. Each of these minimums is of size *k*.

Finally, in the case where non-complementary tuples are partitioned and each partition then processed by the algorithm, each partition is guaranteed to have minimals of size *k* that have combined into connected partitions. Therefore each partition is necessarily larger than *k*.

## 8.4 Comparison to Datafly and μ-Argus

In comparison to Datafly and μ-Argus, the k-Similar algorithm has greater precision because it effectively uses generalization and suppression enforced at the cell level. datafly and μ-Argus generalized values at the attribute level. Datafly suppressed at the tuple level, though μ-Argus suppressed at the cell level. In some cases, Datafly may have less precision than μ-Argus, but Datafly always provides results that are adequately protected. On the other hand, μ-Argus, in some cases, can provide results that do not necessarily adhere to the *k* requirement. In comparison, *k*-Similar provides results that are adequately protected and minimally distorted. On the other hand, Datafly and μ-Argus operate in real-time where *k*-Similar does not.

## 8.5 *k*-Similar as an anonymous data system

*k*-Similar uses the following disclosure limitation techniques: de-identification, equivalence class substitution, generalization, and suppression. Below is a description of the framework in which *k*-Similar operates.

S = {*subjects whose information is included in PT*}
P = *set of all people whose information could possibly be in PT*
PT = *privately held information about S*
*QI = set of attributes with replications in E*
U = P
RT = *k-Similar* (PT)
E = set of publicly available information in today's society
G = set of standard communication methods.
*f = k-Similar*

The system **A**(S, P, PT, *QI*, U, {RT}, E, G, *k-Similar*) is an $\mathbf{ADS_0}$.

Informal proof.
Let PT = data in Figure 34.
There cannot exist fewer than *k* tuples in RT having the same values across *QI* based on the correctness of the *k*-Similar clustering algorithm.
So, *k*-map protection is provided and **A** is an $\mathbf{ADS_0}$.

## 8.6 Future work

1. The *k*-Similar algorithm, unlike Datafly and μ-Argus made use of the value generalization hierarchies to seamlessly integrate generalization and suppression together so as to be treated as one disclosure limitation technique. Incorporate additional disclosure limitation techniques [124] into this approach.

2. The core Datafly algorithm relies on a heuristic to guide its generalization strategy. This heuristic selects the attribute of the quasi identifier having the greater number of distinct values in the modified table as the attribute to generalize. As was discussed earlier [125], this heuristic is computationally efficient but provides no protection against unnecessary generalization. There are many other heuristics that are just as computationally efficient. Develop a nearest neighbor strategy based on distance vectors, like those used in the *k*-similar algorithm, to perform attribute-level generalization and tuple-level suppression that operates in real-time.

3. Construct a more efficient version of *k*-Similar by taking advantage of constraints placed on distances rather than computing the distances of all combinations of *k* tuples. If you do not compute a distance vector between two tuples but have computed the distances of other tuples that include those tuples, then the ⊕ and Θ operations described in Figure 87 can be used to compute the range of possible values for the distance vector between those two tuples.

4. The *k*-Similar algorithm has been described as a data-clustering algorithm that has a symbiotic relationship to the *k*-nearest neighbor algorithm [126]. Compare and contrast these two algorithms as general-purpose data clustering algorithms. Explore ways distance vectors and value generalization hierarchies can be used to improve results in k-nearest neighbor.

5. Earlier in this work, *k*-map, wrong-map and null-map forms of data protection were introduced [127]. These later chapters have been narrowly focused on a version of *k*-

map protection called *k*-anonymity [128]. Explore disclosure control techniques and systems that use other formal protection models.

# Chapter 9 Results: Scrub

Datafly [129], μ-Argus [130], *k*-Similar [131] and even MinGen [132] all work with field-structured data sets. My Scrub system, presented in this chapter, locates personally identifying information in textual documents and within textual fields of a database. This is a change in format from the earlier chapters. As you will see the problem of locating personally identifying information in text can be very difficult, but even when it is resolved perfectly, the results are merely de-identified and not typically rendered anonymous.

## 9.1 Overview of the Scrub System

The Scrub System, which locates and replaces personally identifying information in text documents, textual fields of the database textual information found on the World Wide Web. A close examination of two different computer-based patient record systems, Boston's Children's Hospital [133] and Massachusetts General Hospital [134], quickly revealed that much of the medical content resided in the letters between physicians and in the shorthand notes of clinicians. This is where providers discussed findings, explained current treatment and furnished an overall view of the medical condition of the patient.

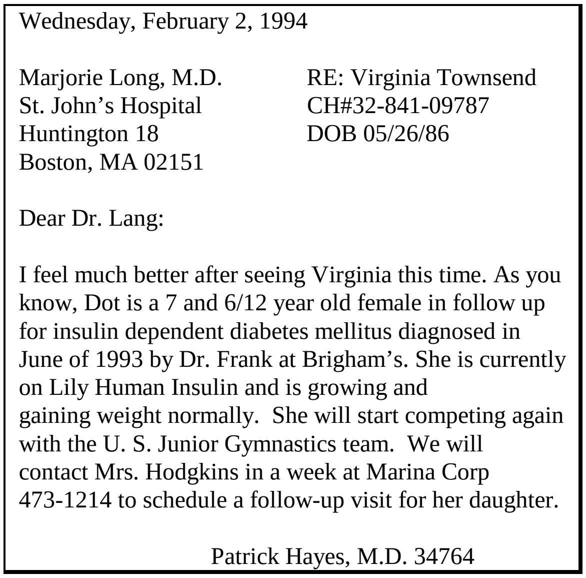
At present, most institutions have few releases of data that include these notes and letters, but new uses for this information is increasing; therefore, the desire to release this text is also increasing. After all, these letters and notes are a valuable research tool and can corroborate the rest of the record. The fields containing the diagnosis, procedure and medication codes when examined alone can be incorrect or misleading. A prominent physician stated at a recent conference that he purposefully places incorrect codes in the diagnosis and procedure fields when such codes would reveal sensitive information about the patient [135]. Similarly, the diagnosis and procedure codes may be up-coded for billing purposes. The General Accounting Office estimates that as much as 10% of annual Federal health care expenditures, including Medicare, are lost to fraudulent provider claims [136]. If these practices become widespread, they will render the administrative medical record useless for clinical research and may already be problematic for retrospective investigation. Clinical notes and letters may prove to be the only reliable artifacts.

The Scrub System provides a methodology for removing personally identifying information in medical writings so that the integrity of the medical information remains intact even though the identity of the patient remains confidential. I term this process "scrubbing". Protecting patient confidentiality in raw text is not as simple as searching for the patient’s name and replacing all occurrences with a pseudo name. References to the patient are often quite obscure; consider for example:

> “…he developed Hodgkins while acting as the U.S. Ambassador to England and was diagnosed by Dr. Frank at Brigham’s.”

Clinicians write text with little regard to word-choice and in many cases without concern to grammar or spelling. While the resulting “unrestricted text” is valuable to understanding the medical condition and treatment of the patient, it poses tremendous difficulty to scrubbing since the text often includes names of other care-takers, family members, employers and nick names.

I examined electronically stored letters written by clinical specialists to the physician who referred the patient. The letter in Figure 107 is a fictitious example modeled after those studied. It contains the name and address of the referring physician, a typing mistake in the salutation line, the patient's nick name, and references to another care-taker, the patient's athletic team, the patient's mother and her mother's employer and phone number. Actual letters are often several pages in length.



Wednesday, February 2, 1994

Marjorie Long, M.D.
St. John’s Hospital
Huntington 18
Boston, MA 02151

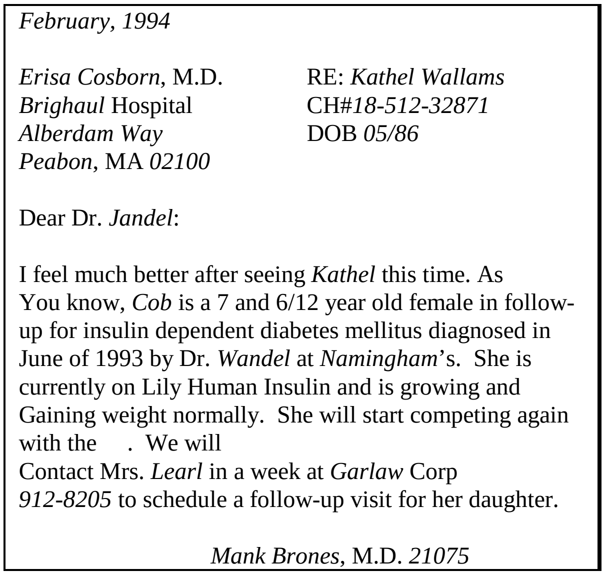
RE: Virginia Townsend
CH#32-841-09787
DOB 05/26/86

Dear Dr. Lang:

I feel much better after seeing Virginia this time. As you know, Dot is a 7 and 6/12 year old female in follow up for insulin dependent diabetes mellitus diagnosed in June of 1993 by Dr. Frank at Brigham’s. She is currently on Lily Human Insulin and is growing and gaining weight normally. She will start competing again with the U. S. Junior Gymnastics team. We will contact Mrs. Hodgkins in a week at Marina Corp 473-1214 to schedule a follow-up visit for her daughter.

Patrick Hayes, M.D. 34764

**Figure 107. Sample letter reporting back to a referring physician.**



*February, 1994*

*Erisa Cosborn*, M.D. RE: *Kathel Wallams*
*Brighaul* Hospital CH#*18-512-32871*
*Alberdam Way* DOB *05/86*
*Peabon*, MA *02100*

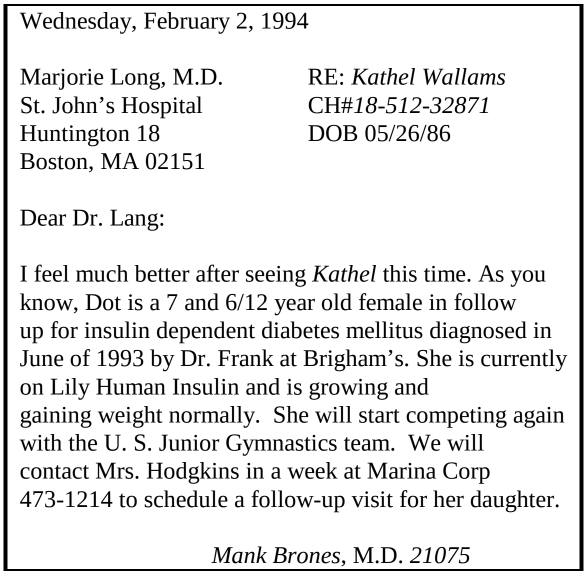
Dear Dr. *Jandel*:

I feel much better after seeing *Kathel* this time. As
You know, *Cob* is a 7 and 6/12 year old female in follow-
up for insulin dependent diabetes mellitus diagnosed in
June of 1993 by Dr. *Wandel* at *Namingham*'s. She is
currently on Lily Human Insulin and is growing and
Gaining weight normally. She will start competing again
with the . We will
Contact Mrs. *Learl* in a week at *Garlaw* Corp
*912-8205* to schedule a follow-up visit for her daughter.

*Mank Brones*, M.D. *21075*

**Figure 108**. **Scrub System applied to sample in Figure 107.**

Figure 107 shows a sample letter and Figure 108 shows its scrubbed result. Notice in the scrubbed result that the name of the medication remained but the mother's last name was correctly replaced. Dates were changed to report only month and year. The reference "U.S. Junior Gymnastics team" was suppressed since Scrub was not sure how to replace it. The traditional approach to scrubbing is straightforward search and replace, which misses these references; this is shown in Figure 109.



Wednesday, February 2, 1994

Marjorie Long, M.D. RE: *Kathel Wallams*
St. John's Hospital CH#*18-512-32871*
Huntington 18 DOB 05/26/86
Boston, MA 02151

Dear Dr. Lang:

I feel much better after seeing *Kathel* this time. As you
know, Dot is a 7 and 6/12 year old female in follow
up for insulin dependent diabetes mellitus diagnosed in
June of 1993 by Dr. Frank at Brigham's. She is currently
on Lily Human Insulin and is growing and
gaining weight normally. She will start competing again
with the U. S. Junior Gymnastics team. We will
contact Mrs. Hodgkins in a week at Marina Corp
473-1214 to schedule a follow-up visit for her daughter.

*Mank Brones*, M.D. *21075*

**Figure 109**. **Search-and Replace applied to sample in Figure 1-8.**

## 9.2 Human approach

The Scrub System was modeled after a human approach to the problem. It uses templates and localized knowledge to recognize personally identifying information. In fact, the work on Scrub shows that the recognition of personally identifying information is strongly linked to the common recording practices of society. For example, Fred and Bill are common first names and Miller and Jones are common last names; knowing these facts makes it easier to recognize them as likely names. Common facts, along with their accompanying templates of use, are considered commonsense knowledge; the itemization and use of commonsense knowledge is the backbone of Scrub.

I conducted an experiment to determine how well humans locate personally-identifying information in letters between physicians. The subjects were 5 adults. None of the subjects had any medical experience or experience with the information contained in the database.

Each of the adults were given a marker that writes in a read-through yellow color and seven (7) printed letters. One of the letters appeared with all its text in uppercase but consisted of complete sentences. The other letters were in standard letter format with upper-lower case. Each subject was asked to highlight all information in each letter that personally identified any person and to do so within 30 seconds per letter.

All the subjects found all obvious references to names, addresses, organizations, cities, states, zip codes and phone numbers (100%). More obscure occurrences such as nick names, abbreviations, identification numbers and incorrect capitalization were sometimes missed (99%). References embedded in the text that did not appear in upper-lower case were sometimes missed (95%) and performance on identifying obvious references in the upper case letter was much worse than in the upper-lower case counterparts (94% compared to 100%). Subjects reported reviewing most words in the letters but all subjects stated they did not read the letters.

I sought to model the human approach because it did not require a complete semantic model. The subjects used templates and localized knowledge to recognize personally identifying information. Consider the list of names, phone numbers and dates in Figure 110. The writing conventions and immediate context help identity the kind of information presented.

| Names | Phone numbers | Dates |
|---|---|---|
| Frank Graves | 255-1423 | March 1, 1991 |
| F. R. Graves, MD | (304) 255-1423 | 3/1/91 |
| Dr. Graves | 304/ 255-1423 | first of March |
| Frank Red Graves | 255-1000 ext 1423 | 1-MAR-91 |
| “Red” Graves | phone: 255-1423 | 03-01-91 |
| frank red graves | extension 1423 | March 1st |

**Figure 110** **Samples of personal information.**

## 9.3 Computer approach

The Scrub System utilizes numerous detection algorithms competing in parallel to label contiguous characters of text as being a proper name, an address block, a phone number, and so forth. Each detection algorithm recognizes a specific kind of information, where recognizable kinds of information can be thought of as fields such as *first name*, *last name*, *street address*, and *date*. There is at least one detection algorithm for each kind of information.

| Scrub | Entities |
|---|---|
| 1. identification block | *{6, 7, 8, 9, 10, 15, 11, 12, 13, 14, 25, 16, 17, 18, 20, 21, 25}* |
| 2. mailing label | *{6, 7, 8, 9, 15, 11, 12, 13, 14, 17, 18}* |
| 3. address block | *{6, 7, 8, 9, 15}* |
| 4. full name | *{11, 12, 13, 14, 17}* |
| 5. location | *{7, 8, 15}* |
| 6. street | 15. country |
| 7. city | 16. social security |
| 8. state | 17. title |
| 9. zip | 18. organization |
| 10. phone | 19. measurement |
| 11. first name | 20. age |
| 12. middle name | 21. date |
| 13. last name | 22. medical term |
| 14. nick name | 25. reference number |

**Figure 111** **Some of the entities recognized by Scrub are listed above in relative order of precedence.**

Figure 111 lists some of the types of entities detected by Scrub. For each entity there is a detection algorithm and the precedence of the algorithm is based on the number of entities that constitute the algorithm’s assigned entity. Examples of constituent entities are listed in braces in Figure 111. For example, detecting a geographical location may make it possible to identify a city, a state or a country. The more constituents an entity has, the higher its precedence. Figure 111 shows five levels of precedence with identification block having the highest and entities 6 through 25 all having the same low precedence.

Detection algorithms can be executed sequentially in order of precedence to avoid parallel execution. For each character in the input text the detection algorithm with the highest precedence reporting the greatest likelihood above a threshold value is considered to have identified an instance of its entity. Figure 112 provides an overview.

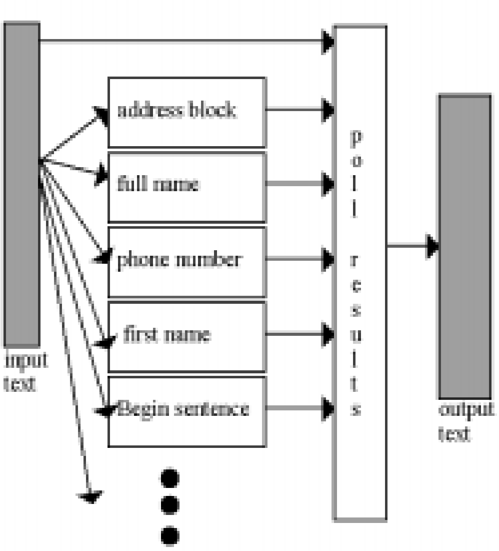


**Figure 112 Block diagram of Scrub detection system.**

Knowing what instances have already been found in the text can be quite useful in reducing ambiguity. For example, if the system encountered the name “Virginia P. Weston” then later encountered a sentence that read “After seeing Virginia this time, I feel much better,” the system could more reliably interpret the second reference to Virginia as a person’s name and not the state. When an instance of an entity is found in the text, its corresponding detection algorithm can post its results -- making them available to all detection algorithms while processing the remainder of the document. In these cases, an entity can only post values for its constituent entities and if there are no constituents, it can post for itself.

A few detection algorithms work differently. Some classify the format of the document as being a letter, notes, or delimited text. These detectors continuously report findings. There are also special detectors like those for medical terms and verbs whose instances are typically not replaced but are detected because having their results reduces the number of false positives. At run-time the user sets the threshold and use of special detectors.

Figure 113 repeats the second column of Figure 110 but includes associated templates and probabilities. The *d* is a digit, the asterisk (*) matches any wild character and the set notation denotes possibilities. During a training session on the database, template probabilities are adjusted and their effectiveness measured. If there is not enough variation between templates then performance will deteriorate. If templates use features that are not present in the database, performance may deteriorate. For example, if name templates expect names to be written in upper-lower case then these templates will be useless if all text appears in uppercase. The training session pinpoints problem areas and weaknesses beforehand.

| Phone numbers | Templates | Likelihood |
|---|---|---|
| 255-1423 | ddd - dddd | 40 |
| (304) 255-1423 | **(** ddd **)** ddd - dddd | 85 |
| 304/ 255-1423 | ddd **/** ddd - dddd | 50 |
| 255-1000 ext 1423 | ddd - dddd ext* d* | 70 |
| extension 1423 | ext* d* | 40 |
| phone: 255-1423 | {tel*, ph*} ddd - dddd | 90 |

**Figure 113 Samples of templates and their probabilities.**

As I've shown, the detection algorithms employ a host of lists. For example, detecting a first name may use a stored list of common first names, the first names of all patients, words that sound like first names or all three depending on the user's specifications. These lists are compiled beforehand. Storage requirements and speed are dramatically reduced using multiple hashed Boolean lookup tables [137] or in the case of words that sound like a group of words, using a table of orthographic rules [138]. With Boolean look-up tables, look-ups are done in constant time, O(10) since there are 10 binary checks per word. Using orthographic rules, look-ups require O(2n) time where n is the number of syllables in the word. Storage using Boolean look-up tables require roughly 30 bits per word which is a tiny fraction of a typical word list or dictionary [139].

### 9.3.1 Replacement Strategies.

Once personally identifying information is detected, it must be replaced with some pseudo-value. There are several strategies for accomplishing this feat. Associated with each detection algorithm in Scrub is a replacement algorithm that is responsible for producing the replacement text; these are the same as was used in Datafly [140]. In general, the format of the replacement text matches the template that was recognized. If the detected entity was a date, for example, the replacement date may involve lumping days to the first of the nearest month or some other grouping. On the other hand if the detected entity was a first name, the typical strategy is to perform a hash-table lookup using the original name as the key. The result of the look-up is the replacement text. This provides consistent replacements since every time a particular name is encountered, it maps to the same replacement. In terms of the replacement content, several other strategies are available including the use of orthographic rules called Sprees[11] that replace personally identifying information with fictitious names that sound like reasonable names but in fact belong to no known person.

## 9.4 Results

The Scrub System accurately found 99-100% of all personally identifying references in more than 3,000 letters between physicians, while the straightforward approach of global search-and-replace properly located no more than 30-60% of all such references; these values are summarized in Figure 114.

The database I used was a scrubbed subset of a pediatric medical record system [141;;-]. It consisted of 275 patient records and included 3,198 letters to referring physicians. Many of the letters were delimited notes but most were proper letters with a heading block, salutation and well-formed sentences.

The higher figure for search and replace (84%) includes using additional information stored in the database to help identify the attending physician's name, identifying number and other information. Since the letters were properly formatted, the heading block was easily detected and compositional cues were available using keywords like "Dear." This dramatically improved the results of the search-and-replace method to around 84%; however, most references to family members, additional phone numbers, nick names and references to the physician receiving the letter were still not detected, whereas Scrub was able to correctly identify and replace these instances.

| **Method** | **Letters** |
|---|---|
| Straight search | 37% |
| Search with cues | 84% |
| Scrub( threshold 0.8) | 98% |
| Scrub( threshold 0.7,<br>false positive reduction) | 100% |

**Figure 114 Comparisons of Scrub to standard techniques**

## 9.5 Discussion

Despite this apparent success, the Scrub System merely de-identifies information and cannot guarantee anonymity. Even though all explicit identifiers such as name, address and phone number are removed or replaced, it may be possible to infer the identify of an individual. Consider the text in 115.

> “At the age of two she was sexually assaulted. At the age of three she set fire to her home. At the age of four her parents divorced. At the age of five she was placed in foster care after stabbing her nursery school teacher with scissors.”

**Figure 115 Sample de-identified text**

If her life continues to progress in this manner, by the age of eight she may be in the news, but nothing in this text required scrubbing even though there would probably exist only one such child with this history. An overall sequence of events can provide a preponderance of details that identify an individual. This is often the case in mental health data, discharge notes and person-specific textual information.

Although Scrub reliably locates explicitly identifying information in textual documents, it merely de-identifies the result because its detectors are aimed primarily at explicitly identifying values. Similarly, in field-structured databases de-identification typically provides insufficient protection, as was demonstrated earlier in this document. Other values remaining in the data can combine uniquely to identify subjects. The Scrub work demonstrates that this is as true in textual documents as it is in field-structured databases. But perhaps more importantly, the Scrub work implies that solving the problem in one data format (either textual documents or field-structured databases) will reveal comparable strategies for solving the problem in the other format.

The Scrub System is both troublesome and insightful in another regard. While Scrub is inadequate for privacy protection, it is quite useful in automatically detecting and gathering personally identifying information from email messages, World Wide Web pages, and other textual information appearing in an electronic format and then using the results to draw damaging inferences from other publicly available field-structured data sets. In this way, Scrub demonstrates the symbiotic relationship between data detective tools and data protection tools. Re-identification experiments and the tools used to accomplish re-identifications improve our understanding of the identifiability of data and our tools for rendering data sufficiently anonymous.

## 9.6 Scrub as an anonymous data system

Scrub uses the following disclosure limitation techniques: de-identification, equivalence class substitution, generalization, and suppression. Below is a description of the framework in which Scrub operates.

S = {*subjects whose information is discussed in textual documents PT*}

P = *set of all people whose information could possibly be PT*

PT = *set of documents about S*

*QI = set of attributes for which Scrub detectors are available*

U = $\{d_1 \times \ldots \times d_n\} \cup P$

RT = *Scrub*(PT)

E = set of publicly available information in today's society

G = set of standard communication methods.

*f = Scrub System*

The system **A**(S, P, PT, *QI*, U, {RT}, E, G, *Scrub*) is <u>not</u> an **ADS$_0$.**

Informal proof.

Assume **A** is an **ADS$_0$.**

Let $p_i$ be the person who is the subject of the text in Figure 115.

E includes newspaper reports and phone books that include $p_i$'s family.

By simply linking the information, $p_i$ can be re-identified, violating property 9 of an **ADS$_0$**.

So, **A** is <u>not</u> an **ADS$_0$.**

# Chapter 10 Discussion

The Scrub System demonstrated that medical data, including textual documents, can be de-identified, but as I have shown, de-identification alone is not sufficient to protect confidentiality. Not only can de-identified information often be re-identified by linking data to other databases, but also releasing too many patient-specific facts can identify individuals. Unless society is proactive, the proliferation of medical data may become so widespread that it will be impossible to release medical data without breaching confidentiality. For example, the existence of rather extensive registers of business establishments in the hands of government agencies, trade associations and firms like Dunn and Bradstreet has virtually ruled out the possibility of releasing database information about businesses [142].

The Datafly, μ-Argus and *k*-Similar systems illustrated that medical information can be generalized so that attributes and combinations of attributes adhere to a minimal *k* requirement, and by so doing, confidentiality can be maintained. Such schemes can provide anonymous data for public use. There are drawbacks to these systems, but the primary shortcomings may be counteracted by policy.

One concern with both μ-Argus, Datafly and *k*-Similar is the determination of the proper value for *k* and its corresponding measure of disclosure risk. There is no standard that can be applied to assure that the final results are adequate. It is customary to measure risk against a specific compromising technique, such as linking to known databases that the data holder assumes the recipient is using. Several researchers have proposed mathematical measures of the risk, which compute the conditional probability of the linker's success [143].

A policy could be mandated that would require the producer of data released for public use to guarantee with a high degree of confidence that no individual within the data can be identified using demographic or semi-public information. Of course, guaranteeing anonymity in data requires a criterion against which to check resulting data and to locate sensitive values. If this is based only on the database itself, the minimum *k* and sampling fractions may be far from optimal and may not reflect the general population. Researchers have developed and tested several methods for estimating the percentage of unique values in the general population based on a smaller database [144]. These methods are based on subsampling techniques and equivalence class structure. In the absence of these techniques, uniqueness in the population based on demographic attributes can be determined using population registers that include patients from the database, such as local census data, voter registration lists, city directories, as well as information from motor vehicle agencies, tax assessors and real estate agencies. To produce an anonymous database, a producer could use population registers to identify sensitive demographic values within a database, and thereby obtain a measure of risk for the release of the data.

The second drawback with the μ-Argus, Datafly and *k*-Similar systems concerns the dichotomy between researcher needs and disclosure risk. If data are explicitly identifiable, the public expects patient permission to be required. If data are released for public use, then the producer must guarantee, with a high degree of confidence, that the identity of any individual cannot be determined using standard and predictable methods and reasonably available data. But when sensitive de-identified, but not necessarily anonymous, data are to be released, the likelihood that an effort will be made to re-identify an individual increases based on the needs of the recipient, so any such recipient has a trust relationship with society and the producer of the data. The recipient should therefore be held accountable.

The Datafly, *k*-Similar and μ-Argus systems quantify this trust by having the data holder identify quasi-identifiers among the attributes requested by the recipient. But recall that the determination of a quais-identifier requires guesswork in identifying attributes on which the recipient could link. Suppose a quasi-identifier is incorrect; that is, the producer misjudges which attributes are sensitive for linking. In this case, the Datafly, *k*-Similar and μ-Argus systems might release data that are less anonymous than what was required by the recipient, and as a result, individuals may be more easily identified. This risk cannot be perfectly resolved by the producer of the data since the producer cannot always know what resources the recipient holds. The obvious demographic attributes, physician identifiers, and billing information attributes can be consistently and reliably protected. However, there are too many sources of semi-public and private information such as pharmacy records, longitudinal studies, financial records, survey responses, occupational lists, and membership lists, to account a priori for all linking possibilities.

What is needed is a contractual arrangement between the recipient and the producer to make the trust explicit and share the risk. Figure 116 contains some guidelines that make it clear which attributes need to be protected against linking. Using this additional knowledge and the techniques presented in the Datafly, *k*-Similar and μ-Argus systems, the producer can best protect the anonymity of patients in data even when sensitive information is released. It is surprising that in most releases of medical data there are no contractual arrangements to limit further dissemination or use of the data. Even in cases where there is an IRB review, no contract usually results. Further, since the harm to individuals can be extreme and irreparable and can occur without the individual’s knowledge, the penalties for abuses must be stringent. Significant sanctions or penalties for improper use or conduct should apply since remedy against abuse lies outside technology and statistical disclosure techniques and resides instead in contracts, laws and policies.

1. There must be a legitimate and important research or administrative purpose served by the release of the data. The recipient must identify and explain which attributes in the database are needed for this purpose.

2. The recipient must be strictly and legally accountable to the producer for the security of the data and must demonstrate adequate security protection.

3. The data must be de-identified. The release must contain no explicit individual identifiers nor should it contain data that would be easily associated with an individual.

4. Of the attributes the recipient requests, the recipient must identify which of these attributes, during the specified lifetime of the data, the recipient could link to other data the recipient will have access to, whether the recipient intends to link to such data or not. The recipient must also identify those attributes for which the recipient will link the data. If such linking identifies patients, then patient consent may be warranted.

5. The data provider should have the opportunity to review any publication of information from the data to insure that no potential disclosures are published.

6. At the conclusion of the project, and no later than some specified date, the recipient must destroy all copies of the data.

7. The recipient must not give, sell, loan, show or disseminate the data to any other parties.

**Figure 116. Contractual requirements for restricted use of data based on federal guidelines and the Datafly System.**

In closing, a few researchers may not find this presentation of the magnitude and scope of the problem surprising, but it has disturbed legislators, scientists and federal agencies [145], so much so, I warn against overreaction especially as it may lead to inappropriate and inoperable policies. I present the problem and these incremental solutions from a belief that knowledge and not ignorance provides the best foundation for good policy. What is needed is a rational set of disclosure principles, which are unlikely to evolve from piecewise reactions to random incidents, but require instead comprehensive analysis of the fundamental issues. The technology described here is quite helpful, but society must still make conscious decisions. There is a danger in over-simplifying this work. It does not advocate giving all the data on all the people without regard to whether individuals can be identified. It does not advocate releasing data that is so general it cannot be useful; substantial suppression does not appear to be the norm. From the viewpoint of the person who is to receive the data, these systems seek to provide the most general data possible that is practically useful. From the viewpoint of privacy, if that level of generality does not provide sufficient protection, then the techniques presented here identify the nature and extent of trust required for a given release of data. Polices and regulations regarding the agreements necessary to make that trust explicit and enforce its terms lie outside the technology.

Consider the case of data released to researchers. When anonymous data is useful, then the data should be released. In some cases completely anonymous data is not practically useful; in those cases, society (and the data holder) can quantify the trust given to researchers who receive more identifiable data. Changes should be made such that public-use files adhere to a reasonably high level of anonymity. In cases where more identifiable data is needed, society should consciously decide how to release such data and the recipient should be held responsible not to violate the contractual agreements that spell out the conditions of trust.

Finally I also warn against doing nothing. Consider an alternative to autonomous database systems, since the burden of determining the risk of disclosure may appear cumbersome. Suppose instead that society had a centralized federal repository for medical data like those found in Iceland and other countries. Though institutions and businesses could maintain their own data for internal purposes, they could not sell or give data away in any form, except of course for disclosure to the federal repository, remuneration for services and required reporting. The recipients of these data would, in turn, be equally restricted against further dissemination. The trusted authority that maintains the central repository would have nearly perfect omniscience and could confidently release data for public use. Questions posed by researchers, administrators and others could be answered without releasing any data; instead the trusted authority would run desired queries against the data and then provide non-compromising results to the investigators.

In releases of de-identified data, the exact risk could be computed and accompanying penalties for abuse incorporated into the dissemination process. While this type of system may have advantages to maintaining confidentiality, it requires a single point of trust or failure. Current societal inclinations suggest that the American public would not trust a sole authority in such a role and would feel safer with distributed, locally controlled data. Ironically, if current trends continue, a handful of independent information brokers may assume this role of the trusted authority anyway. If information brokers do emerge as the primary keepers of medical data (akin to the function that Dunn and Bradstreet serve for business data) they may eventually rank among the most conservative advocates for maintaining confidentiality and limiting dissemination. Their economic survival would hinge on protecting what would be their greatest asset, our medical records.

# Index

# References

1 Kohane et al., “Sharing Electronic Medical Records Across Heterogeneous and Competing Institutions,” in J. Cimino, ed., *Proceedings, American Medical Informatics Association* (Washington, D.C.: Hanley & Belfus, 1996):608-12.

2 Office of Technology Assessment, *Protecting Privacy in Computerized Medical Information* (Washington, D.C.: U.S. Government Printing Office, 1993).

3 See L.O. Gostin et al., “Privacy and Security of Personal Information in a New Health Care System,” *Journal of the American Medical Association*, 270 (1993): at 2487 (citing Louis Harris and Associates, *The Equifax Report on Consumers in the Information Age* (Atlanta: Equifax, 1993)).

4 Louis Harris and Associates, *The Equifax-Harris Consumer Privacy Survey* (Atlanta: Equifax, 1994).

5 G. Cooper et al., “An evaluation of Machine-Learning Methods for Predicting Pneumonia Mortality,” *Artificial Intelligence in Medicine*, 9, no. 2 (1997):107-38.

6 See supra note 1 Kohane et al.

7 B. Woodward, “Patient Privacy in a Computerized World,” *1997 Medical and Health Annual* (Chicago: Encyclopedia Britannica, 1996):256-59.

8 National Association of Health Data Organizations, *A Guide to State-Level Ambulatory Care Data Collection Activities* (Falls Church: National Association of Health Data Organizations, Oct. 1996).

9 P. Clayton et al., National Research Council, *For the Record: Protecting Electronic Health Information* (Washington, D.C.: National Academy Press, 1997).

10 See, for example, Donna E. Shalala, Address at the National Press Club, Washington, D.C. (July 31, 1997).

11 “RMs need to safeguard computerized patient records to protect hospitals,” *Hospital Risk Management*, v9 (September 1993): 129-140.

12 D. Linowes and R. Spencer, “Privacy: The Workplace Issue of the ‘90s,” *John Marshall Law Review*, 23 (1990):591-620.

13 D. Grady, “Hospital Files as Open Book,” *New York Times*, March 12, 1997, at C8.

14 P. Clayton, et al. “For the record: protecting electronic health information,” National Research Council. *(*Washington, DC: National Academy Press, 1997).

15 “Who’s Reading Your Medical Records,” *Consumer Reports*, October (1994): 628-32.

16 Cambridge Voters List Database. *City of Cambridge, Massachusetts*. Cambridge: February 1997.

17 1990 U.S. Census Data, Database C90STF3B. *U.S. Bureau of the Census*. Available at http://venus.census.gov and http://www.census.gov. Washington: 1993.

18 See note 8 National Association of Health Data Organizations.

19 Group Insurance Commission testimony before the Massachusetts Health Care Committee. See *Session of the Joint Committee on Health Care, Massachusetts State Legislature*, (March 19, 1997).

20 Cambridge Voters List Database. *City of Cambridge, Massachusetts*. Cambridge: February 1997.

21 See note 19 Group Insurance Commission.

22 J. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, Rockville, MD. 1988.

23 I. Fellegi. On the question of statistical confidentiality. *Journal of the American Statistical Association*, 1972, pp. 7-18.

24 J. Kim. A method for limiting disclosure of microdata based on random noise and transformation *Proceedings of the Section on Survey Research Methods of the American Statistical Association*, 370-374. 1986.

25 M. Palley and J. Siminoff. Regression methodology based disclosure of a statistical database *Proceedings of the Section on Survey Research Methods of the American Statistical Association* 382-387. 1986.

26 G. Duncan and R. Pearson. Enhancing access to data while protecting confidentiality: prospects for the future. *Statistical Science*, May, as Invited Paper with Discussion. 1991.

27 L. Willenborg and T. De Waal. *Statistical Disclosure Control in Practice*. Springer-Verlag, 1996.

28 G. Duncan and S. Fienberg. Obtaining information while preserving privacy: A Markov perturbation method for tabular data. *Proceedings of Statistical Data Protection 1998*. IOS Press. 1999.

29 A. Hundepool and L. Willenborg. $\mu$- and $\tau$-argus: software for statistical disclosure control. *Third International Seminar on Statistical Confidentiality*. Bled: 1996.

30 T. Su and G. Ozsoyoglu. Controlling FD and MVD inference in multilevel relational database systems. *IEEE Transactions on Knowledge and Data Engineering*, 3:474--485, 1991.

31 M. Morgenstern. Security and Inference in multilevel database and knowledge based systems. *Proc. of the ACM SIGMOD* Conference, pages 357--373, 1987.

32 T. Hinke. Inference aggregation detection in database management systems. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, pages 96-107, Oakland, 1988.

33 T. Lunt. Aggregation and inference: Facts and fallacies. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 102--109, Oakland, CA, May 1989.

34 X. Qian, M. Stickel, P. Karp, T. Lunt, and T. Garvey. Detection and elimination of inference channels in multilevel relational database systems. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, pages 196--205, 1993.

35 T. Garvey, T. Lunt and M. Stickel. Abductive and approximate reasoning models for characterizing inference channels. *IEEE Computer Security Foundations Workshop*, **4**, 1991.

36 D. Denning and T. Lunt. A multilevel relational data model. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, pages 220-234, Oakland, 1987.

37 See supra note 30 Su and Ozsoyoglu.

38 See supra note 31 Morgenstern.

39 See supra note 32 Hinke.

40 J. Hale and S. Shenoi. Catalytic inference analysis: detecting threats due to knowledge discovery. In *Proc. of the IEEE Symposium on Research in Security and Privacy,* pages 188-199, Oakland, 1997.

41 L. Sweeney. Scout: discovering the structure of a database. MIT Artificial Intelligence Laboratory Working Paper. Cambridge: 1995.

42 L. Buczkowski. Database inference controller. *Database Security, III: Status and Prospects*, D.L. Spooner and C.E. Landwehr (eds), Elsevier Science, Amsterdam, pages 311-322, 1990.

43 See supra note 32 Hinke.

44 See supra note 33 Lunt.

45 See supra note 34 Qian, Stickel, Karp, Lunt, and Garvey.

46 D. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

47 D. Denning, P. Denning, and M. Schwartz. The tracker: A threat to statistical database security. *ACM Trans. on Database Systems*, 4(1):76--96, March 1979.

48 G. Duncan and S. Mukherjee. Microdata disclosure limitation in statistical databases: query size and random sample query control. In *Proc. of the 1991 IEEE Symposium on Research in Security and Privacy*, May 20-22, Oakland, California. 1991.

49 C. Skinner and D. Holmes. Modeling population uniqueness. In *Proc. of the International Seminar on Statistical Confidentiality*, pages 175-199, 1992.

50 D. Jensen and H. Goldberg. Artificial intelligence and link analysis. Proccedings from the 1998 AAAI Fall Symposium. American Association for Artificial Intelligence. Orlando, Florida. Menlo Park, CA: AAAI Press, 1998.

51 S. Russell and P. Norvig. *Artificial intelligence: a modern approach.* Englewood Cliffs: Prentice-Hall, 1995.

52 T. Cover and P. Hart. Nearest neighbor pattern classification. In *IEEE Transactions on Information Theory*, 13, 21-27, 1967.

53 R. Duda and P. Hart. *Pattern classification and scene analysis.* New York: John Wiley & Sons, 1973.

54 C. Atkeson, S. Schaal and A. Moore. Locally weighted learning. *AI Review*. 1997.

55 A. Moore and M. Lee. Efficient algorithms for minimizing cross validation error. *Proceedings of the 11th International Conference on Machine Learning*. San Francisco: Morgan Kaufmann, 1994.

56 See supra page 43 De-identification

57 See supra page 56 Suppression

58 See supra page 57 Query restriction

59 See supra page 54 Summary data

[60] T. Dalenius. Finding a needle in a haystack – or identifying anonymous census record. *Journal of Official Statistics*, 2(3):329-336, 1986.

[61] G. Smith. Modeling security-relevant data semantics. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, May 1990.

[62] See supra page 60 Swapping

[63] See supra page 52 Relational database

[64] See supra page 62 Quasi-identifier

[65] See infra page 92 Precision metric

[66] See infra page 107 Datafly

[67] See infra page 125 μ-Argus

[68] See infra page 165 *k*-Similar

[69] See supra page 78 *k*-map

[70] See supra page 107 Datafly

[71] See supra page 125 μ-Argus

[72] See infra page 60 Generalization

[73] See supra page 56 Suppression

[74] Davey, B. and Priestley, H. Introduction to Lattices and Order. Cambridge University Press, 1990.

[75] See supra page 60 Disclosure limitation techniques.

[76] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.

[77] See supra page 61 Usefulness is application specific.

[78] See supra page 88 Example of minimal generalizations.

[79] See supra page 86 Generalized table.

[80] See supra page 80 *k*-anonymity.

[81] See supra page 92 Precision metric.

[82] See supra page 93 *Prec* requirement on domain generalization hierarchies.

[83] See infra page 107 The Datafly II System.

[84] See infra page 125 The μ-Argus System.

[85] See infra page 165 The *k*-Similar alogorithm.

[86] See infra page 195 The Scrub System.

[87] Alexander, L. and Jabine, T. Access to social security microdata files for research and statistical purposes. *Social Security Bulletin.* 1978 (41) No. 8.

[88] See supra page 68 Formal protection models.

[89] See supra page 60 Disclosure limitation techniques.

[90] See supra page 80 *k*-anonymity requirement.

[91] See supra note 16 Cambridge Voter List.

[92] Kohane, I. Getting the data in: three-year experience with a pediatric electronic medical record system. In: Ozbolt J., ed. Proceedings, *Symposium on Computer Applications in Medical Care.* Washington, DC: Hanley & Belfus, Inc, 1994:457-461.

[93] See supra note 19 Group Insurance Commission.

[94] See supra page 97 MinGen algorithm

[95] See supra page 98 MinGen computational complexity.

[96] See supra page 83 Singleton requirement on domain generalization hierarchy.

[97] See supra page 89 Distance vectors and generalization strategies.

[98] See supra page 119 Correctness of the core Datafly algorithm.

[99] See supra page 85 Minimal generalization of a table.

[100] See supra page 92 Minimal distortion of a table.

[101] See supra page 92 Precision metric.

[102] See supra page 121 Datafly heuristic.

[103] See infra page 165 *k*-Similar

[104] A. Hundepool and L. Willenborg, "μ- and τ-Argus: Software for Statistical Disclosure Control," *Third International Seminar on Statistical Confidentiality* (1996) (available at http://www.cbs.nl/sdc/argus1.html.

105 For a presentation of the concepts on which μ-Argus is based, see L. Willenborg and T. De Waal, *Statistical Disclosure Control in Practice* (New York: Springer-Verlag, 1996).

106 L. Sweeney. Inferences from unusual values in statistical data. Carnegie Mellon University, H. John Heinz III School of Public Policy and Management Working Paper. Pittsburgh: 2000. [Abbreviated version forthcoming in a separate publication by Springer-Verlag.]

107 See supra page 81 Lemma regarding *k* occurrences of each value.

108 See supra page 83 Domain generalization hierarchy

109 See supra page 97 MinGen algorithm

110 See supra page 98 MinGen computational complexity.

111 See supra note 27 Willenborg and De Waal.

112 See supra page 80 *k*-anonymity.

113 See supra page 92 Precision metric

114 See supra page 58 *k*-nearest neighbor algorithm

115 See supra page 89 Distance vectors.

116 See supra page 92 Precision metric.

117 See supra page 94 Weighted precision metric.

118 See supra page 89 Distance vectors.

119 See supra page 84 Value generalization hierarchies

120 See supra page 92 Precision metric

121 See supra page 94 Weighted precision metric

122 See supra page 89 Generalization strategies.

123 See supra page 97 MinGen algorithm

124 See supra page 60 Survey of disclosure limitation techniques

125 See supra page 121 Datafly heuristic.

126 See supra page 58 *k*-nearest neighbor algorithm

127 See supra page 68 Formal protection models

128 See supra page 80 *k*-anonymity

129 See supra page 107 Datafly System

130 See supra page 125 μ-Argus System.

131 See supra page 165 *k*-Similar algoirthm

132 See supra page 96 Preferred Minimal generalization algorithm (MinGen)

133 See supra note 92 Kohane.

134 G. Barnett, "The Application of Computer-Based Medical-Record Systems in Ambulatory Practice," *N. Engl. J. Med.*, 310 (1984): 1643-50.

135 Anon., Privacy & Confidentiality: Is It a Privilege of the Past?, Remarks at the Massachusetts Medical Society's Annual Meeting, Boston, Mass. (May 18, 1997).

136 Government Accounting Office, *Fraud and Abuse in Medicare and Medicaid: Stronger Enforcement and Better Management Could Save Billions* (Washington, D.C.: Government Accounting Office, HRD-96-320, June 27, 1996).

137 L. Sweeney, *Multiple hashed binary storage of words -- tiny, fast and almost perfect.* Massachusetts Institute of Technology, AI Laboratory: Working paper. 1996

138 L. Sweeney, *Automatic acquisition of orthographic rules for recognizing and generating spellings.* Massachusetts Institute of Technology, Artificial Intelligence Laboratory: Also MIT Masters Thesis. Working paper. 1996.

139 See note 137 Sweeney, *Multiple hashed binary storage*

140 See supra page 107 Datafly System

141 See note 92 Kohane

142 N. Kirkendall et al., *Report on Statistical Disclosure Limitation Methodology, Statistical Policy Working Paper* (Washington, D.C.: Office of Management and Budget, no. 22, 1994).

143 G. Duncan and D. Lambert, "The Risk of Disclosure for Microdata," *Proceedings of the Bureau of the Census Third Annual Research Conference* (Washington, D.C.: Bureau of the Census, 1987): 263-74.

[144] C. Skinner and D. Holmes, "Modeling Population Uniqueness," *Proceedings of the International Seminar on Statistical Confidentiality* (Dublin: International Statistical Institute, 1992): 175-99.

[145] For example Latanya Sweeney's testimony before the Massachusetts Health Care Committee had a chilling effect on the proceedings that postulated that the release of deidentified medical records provided anonymity. See *Session of the Joint Committee on Health Care, Massachusetts State Legislature*, (Mar. 19, 1997) (testimony of Latanya Sweeney, computer scientist, Massachusetts Institute of Technology). Though the Bureau of the Census has always been concerned with the anonymity of public use files, they began new experiments to measure uniqueness in the population as it relates to public use files. Computer scientists who specialize in database security are re-examining access models in light of these works.