Negative Representations of Information

by

Carlos Fernando Esponda Darlington

B.S., Instituto Tecnológico Autónomo de México, 1995

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

December 2005

Dedication

Para mi Luciernaga preciosa.

Acknowledgments

I would like to thank my Dad, Mom and Sister; Mom, Dad and Sister; Sister, Mom and Dad; Mom, Sister and Dad; Dad, Sister and Mom; Sister, Dad and Mom in no particular order and with all my heart.

I want to thank and acknowledge Stephanie Forrest mentor, collaborator and friend, Paul Helman who taught me a lot and whose input made this work robust, and Elena Ackley for helping me make it real.

Throughout the years the people at the adaptive computation laboratory have listened and provided input, in particular Todd Kaplan, Dennis Chao, Anil Somayaji, Matt Glickman, Josh Karlin, Rob Abbott, Ken Ingham, Hajime Inoue, Gabriela Barrantes, Haixia Jia, Justin Balthrop and Ryan Gerety. Terran Lane, professor and committee member continually poses challenges that enrich my research (thanks). I thank Nitant Kenkre for being in my committee and providing a different perspective on things.

My friends Gerardo, Horacio, Vladimir, Todd, Dennis, Lucinda and Alejandro (Barru) have provided input and support that is impossible to quantify. My sisters Le, Gaby, Susi and Flor have been my strong advocates and have always helped me along the way. In particular, I want to thank Gaby and Xavier for always making my trips back home pleasurable.

Finally, I gratefully acknowledge the support of the National Science Foundation (CCR-0331580, DBI-0309147, CCR-0311686), Defense Advanced Research Projects Agency (grant AGR F30602-00-2-0584), the Intel Corporation, and the Santa Fe Institute. I thank the participants of the PORTIA project for their helpful suggestions and finally, I am very grateful to CONACYT (116691/131686), Fulbright (15992010) and my Mom for their financial support.

Acknowledgments are hard to write since, inevitably, someone important is left out by mistake. I apologize to you.

Ah, and thank you Tomas!

Negative Representations of Information

by

Carlos Fernando Esponda Darlington

ABSTRACT OF DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

December 2005

Negative Representations of Information

by

Carlos Fernando Esponda Darlington

B.S., Instituto Tecnológico Autónomo de México, 1995

Ph.D., Computer Science, University of New Mexico, 2005

Abstract

In this dissertation I present the concept of negative representations of information, discuss some possible implementations, and explore its attributes and applications. The concept is summarized by the phrase "everything except...." What follows—the exceptions—are the negative image of the idea being conveyed: For instance, the statement "I like to eat everything except tofu, mole and key lime pie," defines a person's culinary preferences by explicitly stating what they don't like. In this work I explore the idea from two perspectives and its implications for hiding data. Firstly, I consider the case of the negative representation being an inexact depiction of the positive set, i.e. when not all possible items are characterized. For example, the above gastronomic description does not exhaustively list all of the dishes disliked by the person, as it easy to imagine recipes that nobody would like. I address the question of how to generalize from the given set of items to a likely set, outline a specific scheme, and discuss its computational properties. Secondly, I study the case when we want the negative representation to exactly depict all the items not in the positive set. I show how to efficiently create a compact representation to accomplish this, and discuss the properties of the arrangement.

Several characteristics of describing data negatively are elucidated throughout this work: primarily, that a negative representation can be used to constrain the knowledge gained regarding the positive image, that the amount of information per item is generally lower in a negative representation, and that the way in which answers are inferred using a positive or negative set is fundamentally different. Finally, I outline some operations that take advantage of this change of perspective and that help address some of the privacy concerns of the day.

Contents

List of Figures

Chapter 1

Introduction

Information is and always has been a valuable commodity. It reduces the uncertainty about a specific domain and facilitates the achievement of objectives, from where the wildebeest stops for water (so it can be hunted) to whether a company is about to fire its CEO (so stock can be traded).

With the advent of computers and digital storage devices, the amount of information that is being collected and that can potentially be exploited has grown dramatically. Data is being gathered from every imaginable source and ranges from the scientific (e.g. genome sequences, particle collision streams, ...) to the sociological, where information about individuals such as their demographics, preferences, and spending habits are being amassed. The nature and volume of data pose novel challenges in terms of how it should be used. Central to this, is the question of how it is to be represented, as data representation has immense impact on how it can be utilized.

Take, for instance, the data concerning hourly fluctuations in the price of some stock. Suppose a year's worth of data is at our disposal for analysis. What we can feasibly learn from these data depends at least on whether we have a printout or a digital representation. With the latter, a computer can be readily used to produce aggregate and statistical values, plot graphs and charts, and search the data for patterns, whereas a paper printout will render all such examinations impractical.

The importance of data representation is well understood in the machine learning community [47], consider the case of evolutionary algorithms (natural and artificial) where operators such as one-point crossover come into play—two chromosomes are spliced at a single location, the pieces swapped and pasted to create two new sequences. Representing a gene as a physically contiguous or discontinuous sequence of symbols (nucleotides, bits, etc.) determines which are the possible genomes that can be reached within one generation [81]. A final example is cryptography where the aim is to find data representations from which meaningful information can only be extracted when knowledge of the secret used to create them is provided.

In this dissertation I introduce a novel way to represent data; one in which everything except the items of interest is depicted. Information expressed in this way is referred to as *negative information.* The concept is a familiar one; consider for instance a political speech or an activity report: Omissions are often considered as compelling (substantial) as the items that are actually discussed (the phrase "reading between the lines" suggests there is something being deliberately omitted that is of importance to the discourse). Artists sometimes portray everything but the subject of interest as a means to more meaningfully convey a message [106, 131], and statisticians often use the information of everything but the subset of interest as a more tractable way to compute some value (e.g. one minus the probability of something not happening is the probability that it occurs). An example of special significance, one that inspired the present work, comes from the field of immunology, in particular, the method by which pathogens are identified; the immune system keeps a "negative" image of *self*—the normal constituents of the body—and uses it to recognize foreign material.

Examples are plentiful and ubiquitous, however, in this dissertation I focus on a specific domain, one amenable to rigorous analysis, and study the following: How can information be represented negatively; what are some of the properties of a negative encoding; and, how can these properties be used to our advantage. For instance, you are asked for all the information you have in your address book because there are some specific items a third party wants to verify. You have reservations about distributing your entire directory, and decide to turn in an alternative address book containing all the names, addresses, etc. that are *not* in your “confidential” black book. Strictly speaking you have not been deceitful since both books contain the same information, but how can someone make use of it? What questions can be asked of it in an efficient manner? How voluminous is this so called address book, and how long did it take you to generate it? Finally, let us suppose this complementary address book does not have the same information content and is missing some of the “negative” data. Is it still useful? What can you infer from the answers it provides? In this dissertation I address questions such as these, studying the feasibility of negative representations, drawing out their distinctive properties and discussing their potential benefits for security and privacy.

Chapter 2 reviews related work, specifically in the fields of artificial immune systems, machine learning, databases and areas associated with safeguarding sensitive information. Chapter 3 introduces a scheme inspired by the immune system (IS) in which, given a set of strings S, a surrogate is created that includes S along with some additional strings, yielding an inexact representation of S. The resulting set is precisely characterized and it is shown how it can be represented positively or negatively pointing out some of the tradeoffs between options. Chapter 4 studies how the negative image of a set of strings can be represented exactly, it introduces negative databases as a means to this end, and provides algorithms for creating and updating them efficiently. The chapter examines some distinctive properties of negative databases, in particular, it shows that they are easy to create, easy to query in certain ways, but that it is very hard to derive their entire positive image, making them a natural candidate for certain privacy preserving operations. Chapter 5 investigates negative data representations from an information theoretic standpoint and brings out some additional properties of representing data negatively such as the difference in how queries are answered using the positive or the negative image of a set. At the end of the chapter two schemes are outlined that take advantage of these properties, one for disseminating data and another for collecting it both, in a private fashion. The final chapter concludes the dissertation summarizing the findings and providing avenues for future work.

Chapter 2

Related Work

In this section I will review some of the work done in four major areas that relate to the present work: artificial immune systems (AIS), machine learning (ML), databases, data security and privacy. Other disciplines may come into play, such as information theory and statistics, but here I concentrate on the fields that have tackled related problems and/or that provide significant inspiration.

AIS is reviewed since it has been the primary source of inspiration and because the findings herein will be translated into contributions to that research area. A brief summary of relevant subjects in ML is presented, given that many of the problems posed in AIS have been previously addressed by this community. Furthermore, ML has developed a variety of tools that will prove useful in advancing and understanding the proposed thesis. One goal of this dissertation is to establish a connection between databases and the representations presented herein, for this reason a concise account of some of the relevant database concepts and resources is given. Finally, a review of techniques for safeguarding sensitive data is given to illustrate some of the distinctive characteristics of the schemes proposed herein.

2.1 The Immune System and Artificial Immune Systems

Studying biological systems has often been a recourse for solving engineering problems. Recently (1990's) the immune system (IS) has come under focus for this purpose, giving birth to the field of artificial immune systems (AIS). In what follows I briefly discuss the aspects of the adaptive IS[1] that have served as inspiration to this field, and I provide a brief review of the systems that have resulted from these insights. It is important to note that this is not a complete review of immunology, for which the reader is referred to [84, 121], nor is it a complete account of the AIS field (see [36, 48, 39]).

One of the primary functions, of the Immune System (IS) is to keep the organism healthy, in particular to prevent and defuse illnesses induced by foreign agents known as pathogens. The task of identifying a pathogen is complicated by the possibility of never having encountered it before and by the fact that pathogens are subject to evolution (driven in part to avoid recognition by the IS) and change their form. One strategy thought to be employed by the IS to discriminate between *self*—the normal "components" of the organism—and *nonself*—everything else, including potential pathogens. The theory is known as the self-nonself discrimination paradigm [84, 110] and forms the basis of much AIS related work.

The main focus of the AIS community has been on the adaptive IS of which lymphocytes, T-cell and B-cells, are primary actors. Lymphocytes are distributed throughout the body and are individually capable of binding foreign antigen. The collective actions of lymphocytes ultimately determine whether a bound antigen will be eliminated or not. AIS are thus composed of a collection of agents, that mimic lymphocytes, known as detectors. Additionally a few IS inspired algorithms

[1] In contrast to the innate immune system.

have been proposed that bring these components together, particularly the clonal selection algorithm [38] and the negative selection algorithm [63]. The clonal selection algorithm mimics the way B-cells proliferate upon the encounter of an antigen by creating slightly modified copies of themselves subject to selective pressure. This can be formulated as an evolutionary algorithm with all its benefits and drawbacks [12]. The negative selection algorithm, on the other hand, is modeled after the way T-cells are "trained" not to recognize or bind self molecules. After creation, T-cells migrate to an organ called the thymus where there is a representative sample of the peptides normally found in the host organism. Here, they are subjected to a selective process in which binding of a molecule induces the T-cell to die, known as negative selection. Lymphocytes that survive this and other processes are then released to the rest of the body, with relative confidence that whatever is recognized or bound is in fact a nonself agent. One interesting feature of this algorithm is that it is continually running, perpetually creating new T-cells and dynamically adapting to its environment (although much more active in young people).

One of the most popular applications of AIS has been to intrusion detection [135, 88, 37], owing to the parallel between protecting a computer from intruders and protecting an organism from pathogens. In particular, the work by Hofmeyr et al. [77, 78, 79] introduced a network intrusion detection system that embodies most of the concepts outlined above (namely self-nonself discrimination by a collection of detectors generated by negative selection) along with a few others, and demonstrates the effectiveness of this approach in a dynamic scenario. In addition, AIS have been used for varied applications such as color image classification [116], fault detection [125], recommender systems [29] and even some hardware implementations [20, 21, 22].

Particularly relevant to the present work is the representation of self, nonself and immune cells as binary strings and the use of partial matching to establish detection

[58, 77, 78, 79]. The r-chunks match rule used in this dissertation (Chapter 3), introduced in [13, 55], was inspired by the rule used therein named r-contiguous bits [58, 110, 111], which has been used in many artificial immune system projects. Modeling projects incorporating this rule include [58, 110, 111, 62], and application projects that use it include [63, 35, 78, 88, 20, 21, 22, 120, 11]. Many formal studies of immune algorithms are based on systems employing r-contiguous bits, e.g. [44, 43, 133, 132, 134, 52].

The work proposed here shares with the above approaches the self *vs* nonself distinction, the representation of these sets as fixed length strings, and the use of detectors to recognize members of such sets. However, this work is more concerned with the theoretical aspects and tradeoffs between representing data positively and negatively and with studying the relationship of these representations with database theory.

2.1.1 Distance Measures

In the preceding section I discussed that an immune response may be triggered when an immune cell recognizes (or matches) an antigen. In order to simulate the action of immune cells with digital detectors it is necessary to define a measure that establishes when a match has occurred. This is usually accomplished using a similarity measure that determines how alike the antigen and detector are, and forms the basis from which a match can be declared. The simplest way to envision this idea is to imagine an antigen and a detector as two points on the plane, the length of the line between the two points (the Euclidean distance) can be taken to signify how similar they are. If the distance between them is less than some prespecified value τ, it can be said that the detector recognizes the antigen.

There are a myriad of distance measures that can be used for this purpose besides the ones used in this dissertation (see Chapters 3 and 4). Here is a brief description of some of them:

Hamming Distance One of the most popular measures for establishing the distance between two binary strings is the Hamming Distance, defined as the number of bits which differ between two binary strings. There are several matching functions that extend this notion by assigning different weights to different types of local matches, for example, by only counting when the two strings have a '1' at the same position. Among these are the Russel and Rao, Jaccard and Needham, and the Rogers and Tanimoto similarity functions (see [73] for a good review of these functions).

Edit Distance This measure is also known as the Levenshtein distance and is designated as the smallest number of insertions, deletions, and substitutions required to change one string into another.

Euclidean Distance The straight line distance between two points.

Manhattan The distance between two points measured along axes at right angles.

L_m **Distance** The L_m is the generalized version of the distance between two points. In the plane L_2 is the Euclidean distance and the rectilinear or Manhattan distance is L_1.

2.2 Machine Learning

The problem of distinguishing between two classes of objects—self *vs* nonself—has been widely studied in the machine learning and statistical learning communities, where it is known as binary classification or concept learning [102, 103]. The problem can be stated as follows: Given a collection of data points S with binary labels, find some function that correctly maps each instance to its correct class. It is usually considered that S represents only a sample of possible instances; therefore, it is important for the resulting function to classify novel instances correctly. Some examples of this approach include Nearest-Neighbors [64], neural nets [114] and decision trees [113, 25]. Another way to formulate the problem deals with situations in which only instances of one of the two classes are available. Examples of this approach include single-hypothesis testing [64] and auto-association based classification [86, 85, 72]. This approach is more in line with the paradigm studied in Chapter 3 where, in general, I consider instances of only one class and wish to derive some characterization of the underlying set.

As mentioned in Section 2.1 and further explored in Chapter 3, the proposed system is composed of a collection of detectors whose creation is in some way subject to examples of the target concept. One of the approaches being considered (in the case of positive detection) is one in which each detector carries a subset of the attributes taken from a training instance. The ML community has extensively studied instance-based learning [9], whereby a subset of the training examples is explicitly stored and used to classify novel instances. One interesting property of this approach is that there is no explicit derivation of a target function and membership is determined locally; there is no training of a neural net nor an explicit construction of a function. Among the major algorithms for instance-based learning are nearest-neighbor classifiers [33, 74, 46, 16], and case-based reasoning systems [14, 89, 128].

Non-stationary learning techniques [101, 93, 91] are used whenever the target function must be constantly reassessed. These methods are relevant to the work presented here, in that the system should be flexible enough to promptly modify the definition of what strings are to be regarded as anomalous, in the case of the anomaly detection, or change which items are likely to be present in a database, when such schemes are used for protecting sensitive information.

2.3 Databases

One of the objectives of this work is to construct a connection between IS/AIS and database theory. In the approach taken in Chapter 3 we are presented with a set of feature vectors S that can be viewed as instances of some concept (Section 2.2) or as records in a database. If we choose the latter, we can resort to database theory and design detectors accordingly (see Section 3.3.1). Database theory has studied how, in a relational scheme, a table (a set of records) can be decomposed or broken up into subtables while preserving certain properties [94, 75, 119]. As will be seen in later in Chapter 3, detector creation can be viewed as a decomposition of the observed examples S. Decomposing a table requires knowledge about the semantics of the data such as dependencies among attributes. In the case where such information is not known *a priori*, data mining techniques such as association mining and frequent pattern discovery [6, 7, 40, 130, 71] can provide useful guidance.

Another major goal of the present research is to study the feasibility and properties of representing a database negatively, how regular operations such as updates are to take place and what the query system will look like. The present work introduces algorithms for creating, querying and updating negative databases, as the work progresses into the future more sophisticated operations will be demanded. I expect that insight into this questions will come from database theory itself [10, 107].

2.4 Sensitive Data

The main object of this dissertation is to study the properties of negative representations of data. Storing the negative image of a set rather than the set itself immediately suggests that such a strategy might be useful for protecting sensitive information, being that it is everything but the data we care about which is being stored.

There are several areas of research that are potentially relevant to the ideas discussed in this work regarding the handling of sensitive data—any information that should only be accessible to specific parties or applications. These include: encryption, zero-knowledge sets, privacy-preserving databases, privacy-preserving data-mining, query restriction, secret sharing, and multi-party computation.

An obvious starting point for protecting sensitive data is the large body of work on cryptographic methods, e.g., as described in [117]. Some researchers have investigated how to combine cryptographic methods with databases [60, 59, 18, 129], for example, by encrypting each record with its own key. These techniques, however, are intended to conceal all information about the encrypted data, and it is therefore not appropriate to situations in which some queries should be efficiently supported without revealing the entirety of the records.

Zero-knowledge sets were recently introduced in [99] and provide a primitive for constructing databases that have many of the same properties as those described in Chapter 4, namely, the restriction of queries to simple membership (a similar construction to zero-knowledge sets is presented in [109] in which range queries are possible). However, there are several differences between the two approaches. First, zero-knowledge sets are based on widely believed cryptographically secure methods. Second, zero-knowledge sets require a controlling entity for answering queries. The relaxation of this requirement allows negative databases (see Chapter 4) to perform operations such as set intersection privately and efficiently. Finally, to date, there is no efficient way of updating a zero-knowledge set, while Section 4.4.2 gives efficient algorithms for on-line operations on negative databases—the arrangement introduced in Chapter 4 to represent data negatively.

Negative databases as described in Chapter 4 have the property that it is $\mathcal{NP}$-hard to recover their positive image, i.e. the items of interest. Cryptosystems founded on $\mathcal{NP}$-hard problems [57] have been proposed such as the Merkle-Hellman cryptosystem [98], which is based on the general knapsack problem. These systems rely on a series of tricks to conceal the existence of a "trapdoor" that permits retrieving the hidden information efficiently. However, almost all knapsack cryptosystems have been broken [108], and it has been shown [23, 24] that if breaking such a cryptosystem is $\mathcal{NP}$-hard then $\mathcal{NP}$=$Co\mathcal{NP}$. In general, if a scheme based on a $\mathcal{NP}$-hard result is to be used in a privacy setting it will be necessary to study under what situations it does indeed produce hard to reverse instances and if these instances can be readily obtained. There is a large body of work regarding the issues and techniques involved in generating hard-to-solve $\mathcal{NP}$-complete problems [83, 82, 108, 98] and in particular of SAT instances [100, 32]. The problem of recovering the positive image of a negative database is closely related to solving SAT, where the solutions to the logical formula correspond to the entries in the positive database. Efforts concerned with generating hard instances with specific solutions include [61, 1, 2].

Of particular relevance are one-way functions [67, 105]—functions that are easy to compute but hard to reverse— and one-way accumulators [15, 28] which are similar to one-way hash functions but with the additional property of being commutative. One key distinction between these methods and negative databases is that the output of a one-way function is usually compact and the message it encodes typically has a unique representation. By representing data negatively, as described in Chapter 4, a single message has many possible encodings, an idea that is also exploited by probabilistic encryption [69, 19].

Multi-party computation schemes [136, 68], in which complex operations across databases can be performed privately are related to the applications discussed in this dissertation, in particular when they involve operations such as set intersection. Other approaches to set intersection include [92, 127, 104], where several protocols and data structures are introduced to perform this operation securely and efficiently.

Secret sharing [118] is a technique whereby data is protected by splitting it into several pieces. This primitive is related to the setup of Chapter 5 where the privacy of certain operations relies on negative information being divided into several subsets. Some approaches that rely on secret sharing for protecting databases include [26, 4]. Also relevant to the discussion of Chapter 5 is the area of private information retrieval [31, 66] which focuses on protecting the privacy of the entities consulting the database, rather than the contents of the database itself.

In privacy-preserving data mining, the goal is to protect the confidentiality of individual data while still supporting certain data-mining operations, for example, the computation of aggregate statistical properties [8, 5, 3, 41, 45, 129, 126]. In one example of this approach (Ref. [8]), relevant statistical distributions are preserved, but the details of individual records are obscured. The scheme discussed in Chapter 4 is roughly the reverse of this approach, in that it only supports simple membership queries efficiently. Chapter 5 describes a different use of negative representations that allows for private collection of certain population statistics.

The applications of negative information outlined in Chapter 4 are also related to query restriction [95, 30, 41, 42, 126], where the query language is designed to support only the desired classes of queries. Although query restriction controls access to the data by outside users, it cannot protect an insider with full privileges from inspecting individual records to retrieve information.

There is a large body of work in finding compact representations of a set of binary strings or functions (for example, [87, 112, 97, 27]). My work differs in its need to obtain a compact representation of the complement of the input set without explicitly calculating it, for it may be exponentially larger than its counterpart. However, some of these compact representations may also be useful for describing negative data.

In summary, the existence of sensitive data requires some method for controlling access to individual records. The overall goal is that the contents of a database be available for appropriate analysis and consultation without revealing information inappropriately. Satisfying both requirements usually entails some compromise, such as degrading the detail of the stored information, limiting the power of queries, or database encryption.

Chapter 3

Inexact Representations

As described in Chapter 2 the Immune System (IS) identifies pathogens by making a distinction between *self* and *nonself*, and creating immune cells capable of binding (recognizing) members of nonself. T-cells, a specific type of immune cell, are deployed throughout the body to monitor the well-being of an organism. These cells are equipped with receptors able to bind peptides on the surface of cells; if binding occurs, an immune response may be initiated. T-cells are subjected to a selection process before they are released into the body, which ensures that they recognize only nonself peptides. Even though the complete immune response is much more intricate than this cartoon sketch, the above process exhibits many interesting properties from the perspective of the present work. First, is the fact that T-cells recognize nonself, the notion that instead of recording what is normally occurring in the organism (and presumably good) these cells archive the opposite—a negative image of self. Secondly, that nonself is inexactly depicted, i.e. that there are members of nonself that are not included in its description and will be deemed as self. This imprecision can be interpreted as a generalization from "observed" self to "likely" self, owing to the fact that only a sample of self is available for the immune cell creation process, and to the need of affording plasticity to the definition of self, during the lifetime of the organism.

In this chapter I explore a model for inexact negative representations inspired by the above description of the IS. I focus on an abstract depiction of self and nonself, and restrict the universe of discourse U to finite length strings—a string is understood to be a concatenation of symbols drawn from some finite alphabet. Within this context U is partitioned into two subsets (see Figure 3.1): Real Self (RS) which contains all and only the strings that truly belong to self and Real Nonself (RNS) with all strings not in RS.

Recall from the above discussion that neither RS nor RNS are available at any one time, but rather it is only a subset of RS, denoted by S, that is to be used for creating a model of self (or nonself). There are many possible ways to generalize from a sample to a bigger set, some of which are discussed in Section 2.2. The strategy employed in the current work is to use a surrogate set of strings, referred to as detectors, together with a match rule and a detection scheme to define a subset of U. The match rule determines the strings for which a detector stands and the detection scheme establishes the membership of strings in U (to self or nonself) according to the match rule. The general task can be sketched as follows: Given a set of fixed length strings S (a sample of self items), derive a set of detectors and a matching scheme such that every string in U is classified as either self or nonself. Note that the resulting categorization into self and nonself will not, in general, coincide with what RS and NRS actually are (see Figure 3.1), yet, S must be entirely classified as self. One goal is to create a generalization from S that reasonably approximates this dichotomy.

In this chapter, I set out a framework in which the differences between positive and negative representations can be studied by introducing a match rule and four detection schemes for which the computational requirements, and the ensuing

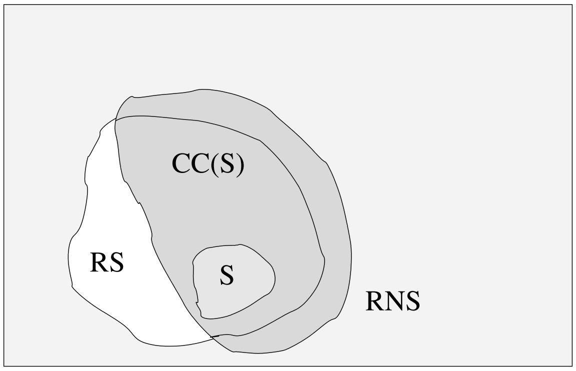


Figure 3.1: The universe of all possible strings is partitioned into two subsets real self (RS) and real nonself (RNS). A subset S of RS is used to create a generalization, denoted by $CC(S)$.

generalizations, are easily characterized[1].

Section 3.1 discusses detectors and the match rule, Section 3.2 outlines the detection schemes and their relationship to one another. The crossover closure is introduced, in Section 3.3, as a generalization construct that can be realized using the match rule and the detection schemes described in the previous sections. The connection of the crossover closure to database theory is also discussed. Section 3.4 presents an analysis of the resulting models in terms of the number of detectors and the size of the generalization for both positive and negative representations.

3.1 Detectors and R-chunks Matching

The main component of the system is a detector. In this section I detail some of the desirable characteristics that a detector should have and discuss the design

[1] The r-chunks match rule outlined in this chapter as well as a description of the detection schemes discussed herein and the database connection have been previously published in [51, 53, 13, 55].

used in this work. The envisioned system is composed of a set of detectors capable of identifying specific properties of the data, and a detection scheme (Section 3.2) which defines the membership of each data item (or string). A detector should be as simple as possible since we want its operation to be efficient in generation time, storage requirements, and run-time (cost of determining a match). Further, the combined characteristics or actions of all detectors (as defined by the detection scheme) should readily define a hypothesis about which strings comprise the underlying concept (self). Recall from above, that the instances or strings from which the model is constructed are represented as attribute vectors or strings. In this context, a natural way to represent a detector is also as a string and to use string matching as the method of detection.

There are several match rules that can been used for this purpose, some of them are described in 2.1.1. In the case of artificial immune systems (AIS) the r-contiguous bits (*rcb*) matching rule, introduced in [110, 111, 77], has been used extensively. An analysis of this rule can be found in [52]. It is from *rcb* that the rule used in this work, *r-chunks*, has been derived. A detector under *rcb* is a string of length l, and is said to match another string, of the same length, if it has at least r consecutive bits in common.

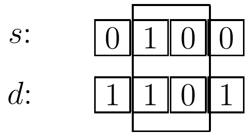


Figure 3.2: Strings d and s match under the *rcb* match rule for r=2.

In r-chunks matching, only r contiguous positions are specified, rather than a sequence of l attributes (usually $r < l$). Thus, an r-chunks detector can be thought of as a string of r bits, together with its starting position within the string, known as its *window*. An r-chunks detector d is said to match a string x if all the symbols of d are equal to the r symbols of x in the window specified by d. More formally, if d is an r-chunk on window w, the matching rule considered is:

$$dMx \leftrightarrow x[w] = d.$$

where dMx denotes that detector d matches string x and $x[w]$ is the projection of string x onto window w. An r-contiguous bit detector can be decomposed into $l-r+1$ overlapping r-chunks detectors, as Figure 3.3 illustrates. As an example, let $d = 1101$ be an r-contiguous bit detector for l=4 and r=2. Let $d[1], d[2], d[3]$ be the corresponding decomposition into r-chunk detectors as illustrated in the following figure:

$$\begin{array}{rl}
d: & \boxed{1}\boxed{1}\boxed{0}\boxed{1} \\
d[1]: & \boxed{1}\boxed{1} \\
d[2]: & \phantom{\boxed{1}}\boxed{1}\boxed{0} \\
d[3]: & \phantom{\boxed{1}\boxed{1}}\boxed{0}\boxed{1}
\end{array}$$

Figure 3.3: r-chunks sliding (overlapping) window decomposition.

Even though the two rules (*rcb* and r-chunks) seem similar at first glance, it was shown in [55] that the r-chunks and the *rcb* match rules are not equivalent in terms of the languages they recognize (see Appendix A).

Notice that r-chunks detectors need not be constructed out of overlapping windows, Section 3.4 outlines other decompositions. In this work I refer to the construction of r-chunks detectors subject to a set of instance strings as a decomposition of the instance strings, in accordance with the database procedure of "decomposing" a table into several tables. Hence the r-chunks matching rule provides the first hint of a connection between this classification system and databases.

It is not the intent of this research to argue that peptide sequences in the immune system can be adequately studied as fixed length strings; nonetheless this simplification, together with r-chunks detectors, makes for a useful analogy with databases. Instance vectors (peptide sequences) may be viewed as entries in a table and a detector model (T-cell receptors) as a particular decomposition of such a table. Database theory has extensively characterized different kinds of decomposition with the purpose of reducing redundancy, improving consistency and facilitating distributivity, among others. It is of interest to understand what questions can be answered about the original data when only queries to detectors are possible; what information is lost, and how compactly it can be represented.

3.2 Detection Schemes

The next step for defining a representation scheme based on string matching is to determine a detection scheme. The match rule establishes which strings a given detector depicts, but says nothing about what a match means, i.e. does a match signify that a string is a member of self or of nonself? Is a single match sufficient to ascertain membership? Etcetera.

In [55], a taxonomy of detection schemes is given in terms of the languages—the set of strings—that a detection scheme is able to recognize[2]. In this section I review these schemes as applied to r-chunks matching. The taxonomy is constructed along two dimensions. The first specifies whether detectors are tailored to match strings in self or nonself, referred to as positive or negative detection, and denoted by P or N accordingly. The second specifies how many matches are required to determine the membership of a string. I will first consider two options for this dimension, one in

[2] The taxonomy was proposed by Paul Helman in [55]. The schemes here referred to as positive conjunctive and negative disjunctive where first discussed in [51].

which a single match suffices and a second in which a detector is required to match in each window, referred to as disjunctive matching (D) and conjunctive matching (C) respectively. I later consider the possibility of intermediate schemes.

Let α denote a choice of P or N and β a choice of C or D. With α and β specified, a collection Υ of r-chunk detectors acts as a parameter, instantiating a detection scheme $Scheme_{\alpha,\beta}(\Upsilon)$. In particular, given the following interpretations for P,N,D,C, $Scheme_{\alpha,\beta}(\Upsilon)$ exactly defines the set of allowable strings—$Scheme_{\alpha,\beta}(\Upsilon)$ protects, or recognizes, this set.

Let U denote the set of all possible strings of length l defined over some alphabet. The language (subset of U) "accepted" by each of the four detection schemes when instantiated with a fixed set Υ of r-chunk detectors is defined as follows:

1. Negative Disjunctive Detection $Scheme_{ND}$: $Scheme_{ND}(\Upsilon)$ is the set of strings x in U such that $(\forall windows\ w)(\nexists d \in \Upsilon)(dMx)$.
2. Positive Disjunctive Detection $Scheme_{PD}$: $Scheme_{PD}(\Upsilon)$ is the set of strings x in U such that $(\exists window\ w)(\exists d \in \Upsilon)(dMx)$.
3. Positive Conjunctive Detection $Scheme_{PC}$: $Scheme_{PC}(\Upsilon)$ is the set of strings x in U such that $(\forall windows\ w)(\exists d \in \Upsilon)(dMx)$.
4. Negative Conjunctive Detection $Scheme_{NC}$: $Scheme_{NC}(\Upsilon)$ is the set of strings x in U such that $(\exists window\ w)(\nexists d \in \Upsilon)(dMx)$.

The following theorem, established by simple set-theoretic arguments, helps clarify the relationships between the classes of languages recognized by the various detection schemes.

Theorem 3.2.0.1. Let Υ be any set of r-chunks detectors and let Υ' denote the complement of Υ relative to the universe of all r-chunks detectors over the same alphabet and of the same length as the detectors in Υ, i.e. all r length strings not in Υ. Similarly, the complement of the set $Scheme_{\alpha,\beta}(\Upsilon)$ (which is a subset of U) is taken relative to U.

Then

$$\begin{aligned} Scheme_{ND}(\Upsilon) &= (Scheme_{PD}(\Upsilon))' = Scheme_{PC}(\Upsilon') = (Scheme_{NC}(\Upsilon'))' \subseteq \\ Scheme_{ND}(\Upsilon')' &= Scheme_{PD}(\Upsilon') = (Scheme_{PC}(\Upsilon))' = Scheme_{NC}(\Upsilon) \end{aligned}$$

Further, the subset containment is proper for some Υ.

It follows from $Scheme_{ND}(\Upsilon) = Scheme_{PC}(\Upsilon')$ that the class of languages recognized by $Scheme_{ND}$ is identical to the class recognized by $Scheme_{PC}$. Similarly, it follows from $Scheme_{PD}(\Upsilon') = Scheme_{NC}(\Upsilon)$ that the class of languages recognized by $Scheme_{PD}$ is identical to the class recognized by $Scheme_{NC}(\Upsilon)$. This result holds for both the r-chunks and the r-contiguous bits match rules.

Above I have considered pairs of schemes that are able to recognize the same sets of strings but that differ in whether negative or positive detectors are used. The choice of a scheme should be weighed by the implementation distinctions they present. In Section 3.4.1, I discuss the tradeoffs in terms of the size of the detector set and the generalization for two r-chunks decompositions. Besides space requirements, the negative representation provides a tantalizing difference: Suppose we have a set of positive detectors generated from some database, each detector then records a potentially meaningful amount of information, for instance an individual's passport number and nationality. On the other hand, if negative detectors are generated, the information contained in each single detector is marginal and difficult to exploit. It is only when all negative detectors are available that the original database (or its closure see Section 3.3) can be reconstructed and individual records inspected (this issue is further explored in Chapter 5). Another interesting difference, pointed out in

[44], between the positive conjunctive and the negative disjunctive schemes is that the latter is inherently distributable; any single detector is capable of classifying a string and can thus act independently. It is one of the tasks of this work to clearly outline the differences between both implementations and their potential applications.

Intermediate schemes

The taxonomy described in the preceding section divides detection schemes according to the number of matches required to determine the membership of a string and considered only the cases where a string must be matched in one or in all of its windows. In this section I briefly discuss the possibility of intermediate schemes. If a metric over the match rule is defined then we can write the intermediate detection scheme as:

$$Scheme_I = \{x | d(x, S) \leq \tau\}$$

where a τ is some threshold and $d(x, S)$ is the distance of string x from the sample instances S. Under an intermediate scheme, a string may be treated as self even if not all its window patterns have been observed before. Intuitively, strings that differ a small amount from the sample are more likely to be a part of the concept than those that differ a lot. Extending this idea one step further, one can imagine distance measures that do not assign a uniform value to every match, but instead take into account structural and statistical properties of the sample in order to weigh the relative merits of distinct matches.

Finally, we can rewrite the original detection schemes in terms of a distance metric. Consider for example $d(x, S)$ to be the number of windows that are present in string x but not in any string in S then:

- $Scheme_{PD}(\Upsilon) = \{x | d(x, S) < t\}$,

- $Scheme_{PC}(\Upsilon) = \{x | d(x, S) = 0\}$,
- $Scheme_{ND}(\Upsilon) = \{x | d(x, S) = 0\}$, and
- $Scheme_{NC}(\Upsilon) = \{x | d(x, S) < t\}$,

where t is the number of windows in x.

The remainder of the chapter focuses on the positive conjunctive and negative disjunctive schemes ($Scheme_{PC}$ and $Scheme_{ND}$) under the r-chunks match rule with the intent to study more closely which strings will be matched by a given set of detectors.

3.3 The Crossover Closure

This section describes a particular construct—the crossover closure—from which a collection of examples, taken from some set (self), can be used to characterize a larger set of instances. The class of sets closed under crossover closure has a central role in this work as I will focus on using the crossover closure as means to generalize from the observed strings.

Consider a sample S of strings taken from a concept class RS (real self) which we wish to characterize. A simple categorical division into “similar to S” versus “dissimilar from S” to distinguish between these categories is by means of a *generation rule* which attempts to characterize the underlying set RS from which S is likely drawn. It has been shown experimentally that this interpretation of the detection task often captures sufficient detail of process behavior to provide effective detection [80, 79].

Definition 3.3.0.1. A generation rule G is a mapping from a set S of length l strings to a set $G(S)$ of length l strings containing S.

I will focus my attention on a generation rule that is both simple to analyze and intuitively appealing, the *crossover closure*. The crossover closure was introduced in [55], where it was restricted to contiguous *windows* of attributes. Here, its definition is extended to a set of features. We understand a feature to be any combination or function of attributes of the strings in U.

Informally stated, given examples of some concept class represented as feature vectors, the crossover closure is a hypothesis stating that some combinations of the observed features define instances of the concept class. To illustrate the basic idea of how this rule operates, consider the simple example of the concept *vehicle*, and the following sample instances (S), where each row of the table corresponds to a single instance:

Wheels	Color	Max. Speed (mph)
4	red	100
2	black	200

Under the crossover operator the following are also valid instances of the class vehicle:

Wheels	Color	Max. Speed (mph)
4	black	100
4	red	200
4	black	200
4	red	100
2	black	100
2	red	200
2	black	200
2	red	100

Formally, given a set S of strings, and a fixed $1 \leq r \leq l$, the crossover closure

$CC(S)$ of S is defined in terms of its features as:

$$CC(S) = \{u \in U | (\forall \textit{features } w)(\exists s \in S)\ u[w] = s[w]\} \tag{3.1}$$

where $u[w]$ is the projection of string u onto feature w. A string u of length l is in the crossover closure of S if and only if each of u's features exactly matches the corresponding feature of some member of S. In the above example, instances of the concept *vehicle* are represented as strings with three features: *Wheels*, *Color* and *Max. Speed*. A string containing any combination of the sampled values for these features is part of the crossover closure. When S is such that $CC(S) = S$ we say that S is *closed* under the crossover operator with respect to a given choice of feature set.

An interesting motivation for this rule stems from the similarities it has with some well known relational database operations [55]. The join operator and the crossover closure are closely related under some partial match rules, leading to the belief that the crossover closure may be a useful characterization for many practical data sets.

The name "crossover closure" was partly inspired by the crossover operation in genetic algorithms (GA) [81]. However, these notions do not exactly correspond, as the crossover discussed here depends on what a feature is defined to be. For the example presented above, in which features are non-overlapping, the crossover closure corresponds exactly to the set of possible strings that can be generated (from an initial population S) using the GA crossover operator alone. However, other decompositions such as the one discussed in Section 3.3.1 do not correspond to the traditional crossover operator. In the overlapping case discussed in Section 3.3.1 , the crossover closure is a proper subset of the possible strings generated by the GA's crossover operator.

There is a strong relationship between the crossover closure and the schemes $Scheme_{ND}$ and $Scheme_{PC}$ when r-chunk detectors are used as shown in [55]. Let S

be any subset of U. Let W_P denote the set of features present in S, that is, the union of the projections of S onto each relevant attribute combination. Let W_N denote the set of features not present in S, that is, $W_N = W_P'$. We then have:

Theorem 3.3.0.2. $Scheme_{PC}(W_P) = Scheme_{ND}(W_P') = Scheme_{ND}(W_N) = CC(S)$.

Proof. $x \in Scheme_{PC}(W_p) \leftrightarrow \forall_w\ x[w] \in W_p \leftrightarrow \forall_w \exists\ s \in S, x[w] = s[w] \leftrightarrow x \in CC(S)$ □

Further, the crossover closure exactly characterizes the class of languages recognized by $Scheme_{PC}$ and $Scheme_{ND}$ when r-chunk detectors are used and fixed definition of feature set is given. That is, we have:

Theorem 3.3.0.3. The class of languages recognized by $Scheme_{PC}$ and $Scheme_{ND}$ when r-chunk detectors are used is exactly the class of sets closed under crossover closure.

Proof. If a set A is closed under crossover closure, i.e. $A = CC(A)$ then we can construct a set of detectors W_P from the features present in A. It follows from theorem 3.3.0.2 that $Scheme_{PD}(W_P) = Scheme_{ND}(W_P') = CC(A)$. By the definition of $Scheme_{PC}$, $x \in Scheme_{PC}(\Upsilon)$ if for all features w $x[w] \in \Upsilon$, where Υ is an arbitrary set of detectors. We construct a set of detectors W_P by taking, for every feature w and every $x \in Scheme_{PC}(\Upsilon)$, the projections $x[w]$ such that $x[w] \in \Upsilon \leftrightarrow x[w] \in W_P$. It follows that $x \in Scheme_{PC}(\Upsilon) \leftrightarrow x \in Scheme_{PC}(W_p)$. Finally, using theorem 3.3.0.2, we have $Scheme_{PC}(\Upsilon) = Scheme_{PC}(W_P) = Scheme_{ND}(W_P') = CC(Scheme_{PC}(\Upsilon))$. □

3.3.1 Database Connection

An interesting analogy can be drawn between relational database theory [65, 94] and decomposing instance strings into sliding window r-chunks(where features are taken to be fixed sized contiguous sequences of attributes [3], see Fig. 3.3). The strings of a sample S can be viewed as the tuples of the current instance of a relation scheme $R(A_1, ..., A_l)$, where each attribute A_i corresponds to string position i and has domain $\{0, 1\}$. For example, the $S = \{0000, 1011\}$ can be represented as a relation scheme $R(A_1, A_2, A_3, A_4)$ whose current instance is shown below:

$R(A_1,$	$A_2,$	$A_3,$	$A_4)$
0	0	0	0
1	0	1	1

For a variety of reasons (e.g., to reduce redundancy and enhance data integrity), it is often advantageous to represent a relation scheme as a decomposition into a collection of smaller relation schemes. Consider representing the scheme $R(A_1, A_2, ..., A_l)$ as a decomposition into schemes $R_1(A_1, ...A_r)$, $R_2(A_2, ..., A_{r+1})$, ... , $R_i(A_i, ...A_{r+i-1})$, ... , $R_t(A_t, ..., A_l)$. The instance of each R_i is the projection of R onto R_i, or equivalently, is exactly the set of strings comprising the i^{th} length r window of S. In the previous example, taking $r = 2$, the instances of the R_i are as follows:

$R_1(A_1,$	$A_2)$
0	0
1	0

$R_2(A_2,$	$A_3)$
0	0
0	1

$R_3(A_3,$	$A_4)$
0	0
1	1

In order to reconstruct the original instance of R from these projections onto the R_i, one computes the natural join of the R_i. However, it is not always the case that

[3] The connection was first suggested by Paul Helman in [55].

the join of the projection recovers the original instance of R—in fact, the join of the projected instances is precisely the crossover closure of the set of tuples (strings) in the original instance of R. In this example, the join $R_1|X|R_2|X|R_3$ of the instances shown above results in the following instance of R, which can be seen to be the crossover closure of the strings in the original instance of R.

$$R_1|X|R_2|X|R_3 = \begin{array}{cccc} R(A_1, & A_2, & A_3, & A_4) \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{array}$$

Relational database theory has exactly characterized when the projections of R join to recover the original instance [65, 94]. This is known as the lossless join condition. The lossless join condition thus also characterizes exactly when a set of strings is equal to its crossover closure with respect to sliding window features. One interpretation of this correspondence is that, since many naturally occurring collections of information do in fact satisfy the lossless join condition, it is plausible that in many contexts the most likely set of strings to be instances pertaining to some concept (RS) is closed under crossover closure. When this is the case, S—if it is a representative sample of instances—can be interpreted as being a sample drawn from $CC(S)$, or from a larger set containing $CC(S)$ and itself closed under crossover. In such situations, it is appropriate to deem strings which are members of $CC(S)$ as relatively likely members of RS.

Some approaches for safeguarding the privacy of records in a database involve deleting certain fields [115, 124] or modifying the values of specific records [8] . It is a conjecture of the present work that the crossover closure can be useful for obfuscating S in a privacy setting by adding "likely" items to a database.

3.4 R-chunks Decompositions

In this section, I discuss two decompositions based on the r-chunks match rule under the $Scheme_{PC}$ and $Scheme_{ND}$ detection schemes described in the previous sections. I examine overlapping and non-overlapping window variants in terms of the size of their detector sets and the size of the generalization they induce (the crossover closure).

3.4.1 Overlapping Fixed Size Windows

The overlapping window decomposition is generated by sliding a fixed length window over the set S of data items (Figure 3.3). This section summarizes the results of the analysis for the $Scheme_{PC}$ and $Scheme_{ND}$ detection schemes. The detailed derivation can be found in Appendices A.2 and A.3.

Given a set of instances S it is straightforward to compute exactly how many distinct detectors can be generated, both for the positive and negative detection schemes [4] ($Scheme_{PC}$ and $Scheme_{ND}$). For $Scheme_{PC}$, it requires counting the number of distinct patterns for each of the $t = l - r + 1$ windows that comprise the strings in S, whereas for $Scheme_{ND}$, enumerating the distinct patterns that are not present in each window will result in the number of detectors. I am interested in how the number of detectors behaves as a function of the window size r and the number of training instances. In the following, I consider the case when S is a random, uniformly generated collection of strings defined over some alphabet of cardinality $\mathbb{A}$. In such a scenario, the expected number of positive detectors E_{pos}

[4] See [43, 133, 134] for a discussion on detector set sizes for the r-contiguous bits match rule.

and the expected number of negative detectors E_{neg} are given by:

$$\begin{aligned} E_{pos} &= tE_r & (3.2) \\ E_{neg} &= t(\mathbb{A}^r - E_r) & (3.3) \end{aligned}$$

where $E_r \approxeq \mathbb{A}^r - \mathbb{A}^r(1 - \mathbb{A}^{-r})^{|S|}$, $t = l - r + 1$ is the number of windows.

With these formulas it is possible to determine when one scheme is beneficial over the other with regards to the number of detectors they require (without attempting to eliminate redundant detectors). For this purpose, it suffices to compute the number of strings in S for which both schemes yield the same number of detectors (i.e., when $E_{pos} = E_{neg}$), and note that a sample smaller than this value will require fewer detectors for the positive scheme, and fewer negative detectors if the sample exceeds it (see Figure 3.4). Both schemes have an equal number of detectors when:

$$|S| \approx (0.693)\mathbb{A}^r \tag{3.4}$$

Note that this value depends only on the choice of r and the cardinality of the alphabet and not on the actual string length (see Appendix A for the detailed derivation).

In a worst case scenario, $Scheme_{PC}$ may require $t\mathbb{A}^r$ detectors when $|S| \geq \mathbb{A}^r$. Similarly, $Scheme_{ND}$ can also yield up to $t\mathbb{A}^r$ detectors but only when there are no self strings whatsoever. It is worthwhile mentioning that, for a given data set S, the number of negative detectors grows exponentially with the size of r. Chapter 4 introduces a representation that allows a detector set to be stored efficiently and algorithms to keep it up to date

A detector set generated in this manner is likely to be redundant. That is, it will likely contain some detectors whose removal would not change the language recognized. Redundancy might be a desirable feature, especially if detectors are to be

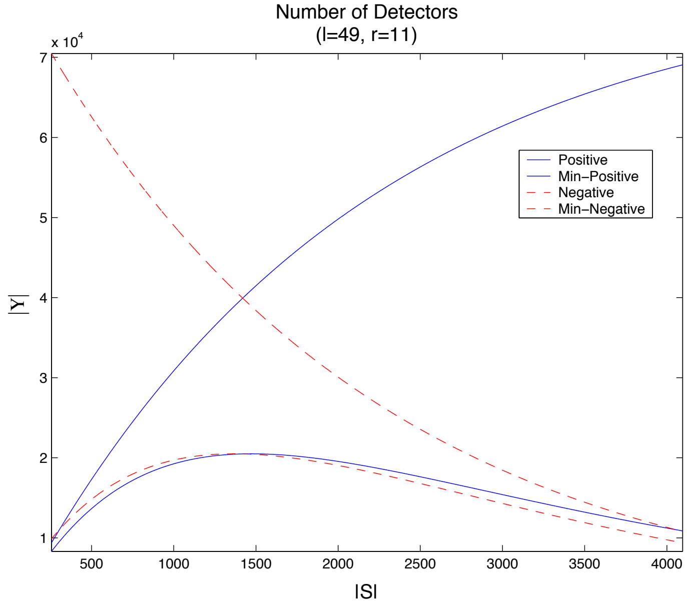


Figure 3.4: Number of positive and negative detectors as a function of the size of the self set, assuming the self set was generated randomly. The plot shows both complete and reduced detector sets.

distributed. However, it is important to understand how much redundancy is present and address the related question of finding an efficient detector set. Intuitively, redundancy arises from the fact that for some strings a match in window i implies a match in window $i+1$. For example, for $l = 3$ and $r = 2$ consider the following sample $S = \{000, 101\}$. The implied generalization consists of strings $CC(S) = \{000, 101, 001, 100\}$. Clearly, the positive detectors for window 1, $\{00, 10\}$, match every string in the closure, and only such strings. Thus, it is unnecessary to check for a match in window 2. It is easily verified that the same holds if negative detectors are used, the case most relevant to AIS. The following two equations eliminate this source of redundancy from the detector set when a binary alphabet is used:

$$E_{minN} = 2^r - E_r + (l-r)(E_r - 2(E_r - E_{r-1})) \tag{3.5}$$

$$E_{minP} = E_r + (l-r)(E_r - 2(E_r - E_{r-1})) \tag{3.6}$$

The size of the sample S for which E_{minN} and E_{minP} yield the same number of detectors is the same as with the full repertoire (see eq.3.4). One subtlety about this analysis is that if not all positive detectors are represented explicitly, then some additional information is required to identify implicit matches, due to the conjunctive nature of the detection scheme. This extra information is unnecessary if negative detection is used and points to an interesting characteristic of using a negative representation, namely, that it is inherently distributable (see Chapter 5 and Ref. [44]). A plot of E_{minP} and E_{minN} for different sample sizes is presented in Figure 3.4 without regards to the extra information required to determine implicit matches.

3.4.2 Crossover Closure and its Expected Size

In order to examine the size of the crossover closure as a function of the sample size $|S|$ and window size r, a similar assumption is made as in the preceding section, namely we consider the case of a random sample of strings over a binary alphabet. This problem can be mapped into a graphical representation, a directed acyclic graph (DAG), where each level contains nodes corresponding to the positive detectors derived from S for each of its windows. Nodes in consecutive levels are connected if the detectors to which they correspond overlap (crossover), i.e. this is if they match in their common $r-1$ positions. Take, for instance, a self set S comprised of the following two strings $S = \{0101, 1111\}$ with $l = 4$, $r = 2$.

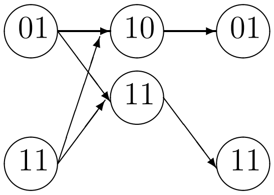


Figure 3.5: DAG representation of the crossover closure.

Under this representation, the crossover closure is exactly the set of strings formed by traversing the graph from level 1 to level t (the number of windows): $CC(S) = \{0101,$ $1111,\ 0111,\ 1101\}$.

In order to compute the number of strings in the closure, observe from the DAG that the number of paths departing from a given node doubles if it has two outgoing edges, i.e., if the corresponding detector crosses-over with two detectors in the following window. The closed form solution for the expected number of paths in such a graph and hence the size of the crossover closure is given by:

$$CC(t) = E_r(\bar{o})^{(t-1)} \tag{3.7}$$

where E_r is the expected number of positive detectors given by eq. 3.2, $\bar{o}$ is the expected out degree of each node and t is the number of windows, see Appendix A for the detailed equation.

The size of the generalization, $CC(S)$, can be determined by the size of S or by the size of the detector set (obtaining an estimate for $|S|$ from either eq. 3.3 or eq. 3.2 and substituting in eq. 3.7). It is important to note that the actual size will depend on the structure of the specific self set. Nevertheless, the analysis provides insight into the behavior of the crossover closure, and elucidates the impact of allowing novel strings into the sample. This can be useful for determining, in a dynamic scenario, when (or at what rate) detectors should be added or deleted from the working set.

3.4.3 Non-overlapping Fixed Size Windows

As noted in Section 3.1, the r-chunks match rule does not necessarily require the use of a sliding window nor is it restricted to a fixed window size. As an illustration of another matching scheme for which the crossover closure is the characteristic generalization, I discuss an alternative decomposition based on the r-chunks match rule.

This decomposition restricts detector creation to non-overlapping windows but maintains a fixed size for each window. Assume, for simplicity, that r exactly divides l. In the case of positive detection, all distinct patterns in non-overlapping windows become potential detectors (see Figure 3.6). Alternatively, all patterns absent from such windows, become detectors when a negative scheme is considered.

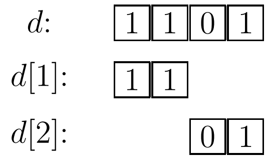


Figure 3.6: Non-overlapping decomposition of string d into positive detectors $d[1]$ and $d[2]$.

It is straightforward to see that the generalization implied by this design is the crossover closure of the strings in the training sample S (according to eq. 3.1). In fact its size is easier to compute than with the sliding window model and is simply the product of the number of detectors for each window:

$$CC(S) = \prod_i |\Upsilon[i]| \tag{3.8}$$

where $\Upsilon[i]$ is the set of detectors for the ith window.

Likewise the number of detectors can be obtained as the sum of the number of detectors for each window:

$$\sum_i |\Upsilon[i]| \ . \tag{3.9}$$

If we consider, as we did above, a random sample of strings S, the expected number of detectors is simply tE_r for positive detectors and $t(\mathbb{A}^r - E_r)$ for negative detectors,

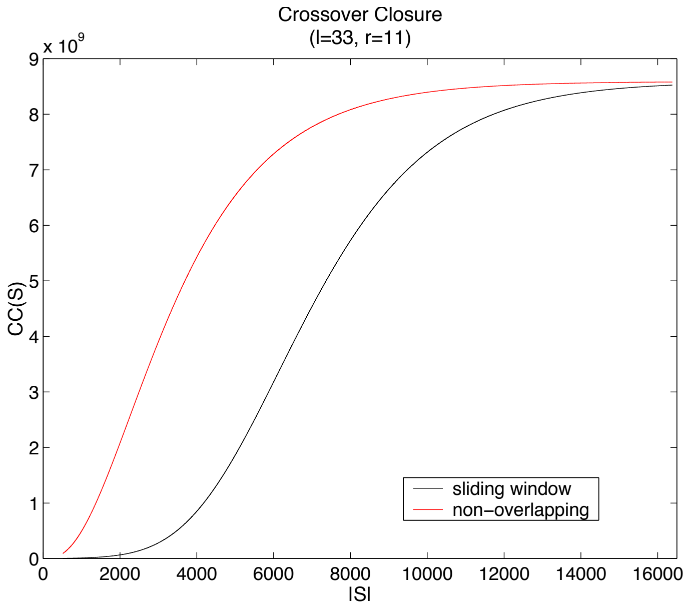


Figure 3.7: The crossover closure as a function of the size of a random sample of binary strings for both the sliding window and non-overlapping decompositions.

where E_r is given by (eq. 3.2) and $t = \frac{l}{r}$ is the number of windows. Interestingly, the tradeoff point between negative and positive detectors is the same as with the sliding window model (eq. 3.4). Lastly, the number of strings encompassed in the generalization follows directly from (eq. 3.8): $CC_r(S) = (E_r)^t$. Figure 3.7 plots the size of the crossover closure of the two decompositions reviewed in this section.

Consider a final decomposition, one in which $r = l$. In this case the the size of the crossover closure is precisely the same as the size of S, i.e. $CC(S) = |S|$ and the detector set exactly distinguishes S from $U - S$. One concern that immediately comes to mind is that the size of the negative detector set is, in general, exponentially bigger than S, and as such infeasible to generate. Chapter 4 addresses this issue by introducing a match rule that allows for an exact and efficient depiction of $U - S$, and discusses some interesting properties and applications.

3.5 Summary

Proper data representation is fundamental to any enterprise wishing to exploit it, be it the modeling of the underlying processes that generated it, mining the data for associations or simply storing it for future reference. In this chapter I have introduced a means to represent data imprecisely; given a set of data items S, a superset of S, named the crossover closure of S, is created that partitions the universe U of possible strings into two subsets, self and nonself. The resulting model consists of a set of strings, called detectors, a match rule, named r-chunks, and a detection scheme. Each detector stands for several strings as defined by the match rule, and the detection scheme determines whether the matched strings belong to self or nonself. I have characterized exactly which strings each partition has under the model, and, being that either of the sets is enough to decide the membership of any string in U, have shown some tradeoffs of representing U using one set or the other. I allude to the contents of the subset containing S as positive information and to the rest as negative information, the act of classifying strings using these sets is referred to as positive and negative detection respectively. Some connections between the proposed scheme and database theory were established and enable the exchange of tools and theoretical results. In the chapters that follow, I discuss how to create an exact representation of $U - S$ (in contrast with the inexact schemes of this chapter) and discuss further distinctions between keeping negative and positive information.

Other work extending the analysis of this match rule includes [122, 70] and applications to anomaly detection are discussed in [13, 123, 90].

Chapter 4

Exact Representations

Generally speaking, when we think of storing data we understand that it will be exactly kept in the database—with no items omitted or added to it. Chapter 3 discusses some scenarios in which inexact representations would be useful and outlines some possible avenues to do so. Central to that discussion is the notion of representing data negatively. In the present chapter I further delve into this idea by investigating how to store negative data exactly and study the properties of the resulting representation.

In this approach [1], the negative image of a set of data records is represented rather than the records themselves (Figure 4.2). The data in question consists of finite-length strings (or records), all of the same length l, and defined over a binary alphabet. The set of all possible strings of this form, the universe U, is logically partitioned into two disjoint subsets: DB representing the set of records that hold the information of interest, and $U - DB$ denoting the set of all strings not in DB (referred to sometimes as *self* and *nonself* respectively). I assume that DB is uncompressed (each record is represented explicitly), but allow $U - DB$ to be stored in a compressed

[1] Some of this work has been previously published in [54, 49, 50].

form called NDB. DB is referred to as the *positive* database and NDB as the *negative* database.

From a logical point of view, either database will suffice to answer questions regarding DB. However, the different representations present distinct advantages. For instance, in a positive database, inspection of a single record provides meaningful information. However, inspection of a single (negative) record reveals little information about the contents of the original database. Because the positive tuples are never stored explicitly, a negative representation would be much more difficult to misuse. Similarly, depending on the specific representation of NDB, the efficiency of certain kinds of queries may be significantly different than the efficiency of the same query under DB. Some applications may benefit from this change of perspective. Most applications seek to retrieve information about DB as efficiently and accurately as possible, and they typically are not explicitly concerned with $U - DB$. Yet, in situations where privacy is a concern it may be useful to adopt a scheme in which certain queries are efficient and others are provably inefficient. Consider, for example, a watch list that is available to airline agents. It is desirable for these agents to have the ability to verify whether a given name is on the list, but at the same time not to have the ability to arbitrarily browse its contents (or even assess its size), lest it fall into the wrong hands. Or imagine the need to privately determine the intersection of sets owned by different parties. For instance, two or more entities might wish to determine which of a set of possible "items" (e.g. transactions) they have in common without revealing the totality of the contents of their database or its cardinality. A longer term motivation concerns a large database of personal records, which an outside entity might need to search, for example, to identify suspicious activities or to conduct epidemiological studies. Under this scenario, it is desirable that the database support only the legitimate queries while protecting the privacy of individual records, say from inspection by an insider.

This chapter presents some initial results on the feasibility of the negative database scheme and illustrates how alternative negative database representations can produce distinct properties with respect to retrieving information or protecting privacy. In the following sections, I first show that implementing NDB is computationally feasible by introducing a scheme that requires $O(ln)$ negative records to represent the complement of a positive database consisting of n l-bit strings, and then giving an algorithm for finding such a representation efficiently. I then investigate some of the privacy implications of the negative scheme. In particular, I show that the general problem of recovering a positive database from the negative representation is $\mathcal{NP}$-hard. I then present a randomized algorithm for creating negative representations that are difficult to reverse, as well as operations for updating and maintaining negative databases. I also study what types of queries can be carried out efficiently under this representation and how negative databases can be used to perform set intersection—an important operation among databases. Finally, I discuss the potential consequences of the results, and outline areas of future investigation.

4.1 Representation

In order to create a database NDB that is reasonable in size, it is necessary to compress the information contained in $U - DB$. To this end an additional symbol is introduced, known as a "don't care," written as $*$. The entries in NDB will thus be l-length strings over the alphabet $\{0, 1, *\}$. The don't-care symbol has the usual interpretation, matching either a one or a zero at the bit position where the $*$ appears. Positions in a string that are set either to one or zero are referred to as "defined" or "specified" positions, and locations where a $*$ appears are referred to as "unspecified" positions. With this new symbol large subsets of $U - DB$ can be represented with just a few entries.

For example, the set of strings $U - DB$ can be exactly represented by the NDB set shown below:

DB	$(U - DB)$		NDB
	001		
000	010		0*1
111	011	$\Rightarrow$	*10
	100		10*
	101		
	110		

The convention is that a binary string s is taken to be in DB if and only if s fails to match each of the entries in NDB. This condition is fulfilled only if for every string $t_j \in NDB$, s disagrees with t_j in at least one defined position.

4.1.1 The Prefix Algorithm

In this section I present an algorithm as proof that a negative database NDB can be constructed in reasonable time and of reasonable size. The prefix algorithm introduced here is deterministic and reversible, which has consequences for the kinds of inferences that can be made efficiently from NDB. We would like some inferences to be hard (e.g., inferring the original DB from NDB) and other inferences to be easy, depending on the application (e.g., finding certain kinds of correlations in DB). However, in the current work, I focus on the question of how easy it is to recover the original DB from NDB, a question addressed in Section 4.2, and on the types of queries supported by the scheme. An example DB, $U - DB$ and the NDB produced by the prefix algorithm is given in Figure 4.2.

Lemma 4.1.1.1. The prefix algorithm creates a database NDB that matches exactly those strings not in DB.

```
Prefix algorithm
For all $i$, let $w_i$ denote an $i$-bit prefix and
$W_i$ a set of $i$-length bit patterns.
1.   $i \leftarrow 0$
2.   Set $W_i$ to the empty set
     Repeat
3.       Set $W_{i+1}$ to every pattern not present in
         $DB$'s $w_{i+1}$ but with prefix in $W_i$
4.       for each pattern $V_p$ in $W_{i+1}\{$
5.           Create a record using $V_p$ as its prefix
             with the remaining positions set to *
6.           Add record to $NDB$.$\}$
7.       Increment $i$ by one
8.       Set $W_i$ to every pattern in $DB$'s $w_i$
9.   Until $i = l$.
```

Figure 4.1: The Prefix algorithm outputs a negative database NDB of size $O(l \cdot |DB|)$ representing the strings in $U - DB$. See Figure 4.2 for an example input/output of the Prefix algorithm.

Proof. Every string not in DB must have a minimum length prefix that is not a prefix of any string in DB. Step three of the algorithm (Figure 4.1) finds these prefixes and, for every such prefix, it appends a representation of every possible string with that prefix to NDB (step five). If a pattern is not present in DB's window w_{i+1} and its own prefix is not in w_i then it must have been inserted in NDB before. Step two initializes W_0 so that the first iteration considers every pattern absent from DB. □

Theorem 4.1.1.1. The negative data set $(U - DB)$ can be represented using $O(l \cdot |DB|)$ records.

Proof. For every window of size i there are at most $|DB|$ "negative" records created and inserted in NDB (steps 4–6). The number of windows is at most l (step 9) therefore, the number of negative records is $O(l \cdot |DB|)$. □

The NDB produced by the prefix algorithm has some interesting properties. For example, each string in $U - DB$ is matched by exactly one NDB record. This non-overlapping property allows NDB to support more powerful queries than simple membership, as I show in Section 4.3.1. For example, it is easy to determine if a given subset of U is present or not in NDB and to what extent; suppose we want to establish how many strings with the first n bits set to one are represented in NDB, then selecting all NDB entries with the first n bits set to one and adding the number of strings each of these entries represents will yield the correct answer.

DB	$U - DB$	NDB	c-keys	$RNDB$
0001	0000	11**	11**	11**
0100	0010	001*	0*1*	0*1*
1000	0011	011*	*11*	1110
1011	0101	0000	00*0	*111
	0110	0101	*1*1	00*0
	0111	1001	1*01	*1*1
	1001	1010	**10	0101
	1010			1*01
	1100			**10
	1101			*010
	1110			
	1111			

Figure 4.2: Column 1 gives an example DB, column 2 gives the corresponding $U - DB$, column 3 gives the corresponding NDB generated by the prefix algorithm,, column 4 presents some possible c-keys extracted from NDB, and column 5 gives an example output of $RNDB$(see Section 4.4).

4.2 Reversibility

In Section 4.1.1, I presented an algorithm for generating NDB that demonstrates the feasibility of a negative representation. I now turn to one property of negative representations. In Ref. [54, 56] we establish that the representation described in Section

4.1 is potentially difficult to reverse i.e. difficult to recover the original DB, and in Section 4.4 present an algorithm aimed at producing hard to reverse instances, I now review those results. It is important to clarify that establishing whether an instance, or class of instances, is truly difficult to reverse remains a empirical endeavor. There is a yearly competition for solving SAT formulas (a problem isomorphic to reversing a negative database (Figure 4.3)) that put the most powerful algorithms to the task, yet some formulas remain unsolved. On the other hand, it is straightforward to devise an efficient method for reversing the outputs of the Prefix algorithm (Figure 4.1). In this section I explain how the representation introduced in Section 4.1 yields databases that might be difficult to reverse, but not all such databases have this property.

Reconstruction of DB from NDB is $\mathcal{NP}$-hard in the following sense [2].

Definition 4.2.0.1. Self Recognition (SR):
INPUT: A set NDB of l-length strings defined over the $\{0, 1, *\}$ alphabet , and a candidate self set DB.
QUESTION: Does NDB represent the self set DB?

We establish that SR is $\mathcal{NP}$-hard. Note that NDB represents an arbitrary set $U - DB$, and we do not specify how it was obtained. First we establish the $\mathcal{NP}$-completeness of the following problem.

Definition 4.2.0.2. Non-empty Self Recognition (NESR):
INPUT: A set $U - DB$ of binary strings represented by a collection NDB of length l strings over the alphabet $\{0, 1, *\}$.
QUESTION: Is DB nonempty? That is, is there some string in $U = \{0, 1\}^l$ not matched by NDB?

Theorem 4.2.0.2. NESR is $\mathcal{NP}$-complete.

[2] This proof was first outlined by Paul Helman in [54].

Proof. NESR is clearly in $\mathcal{NP}$. (If we guess a string, it is easy to verify that it is not matched, and thus a member of DB, by comparing it against every record in NDB.)

The $\mathcal{NP}$-completeness of NESR is established by transformation from 3-SAT. Start with instance $\mathbf{I}$ of 3-SAT. Let X be the set of variables $\{x_i\}$, and suppose l is the number of variables. The constructed instance of NESR will be over length l strings. Each clause $\{L_i, L_j, L_k\}$ in $\mathbf{I}$ (L_i is a literal, which is either x_i or x_i complement) creates a length l string in NDB as follows. All positions other than i, j, or k contain $*$. Position i contains 0 if L_i is x_i and contains 1 if L_i is $\bar{x_i}$ (complemented x_i). A similar construction is used for the other two literals L_j and L_k in this clause. Figure 4.3 shows an example of this mapping.

Claim: There exists a truth assignment satisfying $\mathbf{I}$ if and only if there exists a string in $U = \{0,1\}^l$ not matched by NDB (and therefore in DB). In the following, if $\mathcal{A}$ is a truth assignment to the variables in X, $S(\mathcal{A})$ is the string in U obtained by setting the i^{th} bit to 1 if $\mathcal{A}$ assigns $x_i = T$ and the i^{th} bit to 0 if $\mathcal{A}$ assigns $x_i = F$.

We have:

$\mathcal{A}$ satisfies $\mathbf{I}$

$\iff$ for every clause $C_q = \{L_i, L_j, L_k\}$, at least one literal is satisfied

$\iff$ $S(\mathcal{A})$ fails to match at least one of the bits i, j, k of the q^{th} member of NDB (generated from C_q), because uncomplemented literal L_i generates 0 in the i^{th} position and complemented L_i generates 1 in i^{th} position, and similarly for L_j, L_k

$\iff$ $S(\mathcal{A})$ is in DB.

□

Corollary 4.2.0.1. NESR is $\mathcal{NP}$-complete even if every record of NDB contains exactly three defined positions.

Proof. This transformation always produces such an instance of NESR. □

Corollary 4.2.0.2. Empty Self Recognition (ESR, the complement of NESR, answers YES if and only if NDB represents the empty set) is $\mathcal{NP}$-hard.

Proof. Trivial Turing transformation from NESR. □

Theorem 4.2.0.3. Self Recognition (SR, defined above) is $\mathcal{NP}$-hard.

Proof. We have established this to be the case even when the candidate self set DB is empty, and even when every member of NDB contains exactly three defined positions.

□

Boolean Formula	NDB
(x_1 or x_2 or $\bar{x_5}$) and	00**1
($\bar{x_2}$ or x_3 or x_5) and	*10*0
(x_2 or $\bar{x_4}$ or $\bar{x_5}$) and $\Rightarrow$	*0*11
($\bar{x_1}$ or $\bar{x_3}$ or x_4)	1*10*

Figure 4.3: Mapping SAT to NDB: In this example the boolean formula is written in conjunctive normal form (CNF) and is defined over five variables $\{x_1, x_2, x_3, x_4, x_5\}$. The formula is mapped to an NDB where each clause corresponds to a record and each variable in the clause is represented as a 1 if it appears negated, as a 0 if it appears un-negated and as a $*$ if it does not appear in the clause at all. It is easy to see that a satisfying assignment of the formula such as $\{x_1=$ FALSE, $x_2=$ TRUE, $x_3=$ TRUE, $x_4=$ FALSE, $x_5=$ FALSE $\}$ corresponding to string 01100 is *not* represented in NDB and is therefore a member of DB.

4.3 Applications

4.3.1 Queries

Using the representation described above, negative databases consist of a set of strings defined over $\{0, 1, *\}^l$. Queries to such databases are also expressed as strings defined over the same alphabet and have the form "Is Q a member of DB?" I refer to these queries as authentication or simple membership queries. Queries such as "Which are the engineers in DB?" can be constructed using simple membership queries, as is briefly discussed below.

If string Q consists only of defined positions, i.e. it has no don't care symbols, then determining membership is straightforward as it requires only to ascertain if Q is matched by any one of the strings in NDB (matching is described in Section 4.1). On the other hand, if Q contains an arbitrary number of unspecified positions, answering the query is equivalent to asking whether the corresponding SAT formula has any satisfying assignments when an arbitrary number of its variables have pre-assigned truth values. This remains an $\mathcal{NP}$-hard problem for arbitrary sets of pre-assigned truth values. This contrasts with a positive database DB, where the records are stored explicitly and answering such queries takes time proportional to the size of DB.

For example, consider a database DB of the tuples <name, address, profession> and the query Q "Which are the engineers in DB?", which can be written as a string over $\{0, 1, *\}$, where the profession field is set to the binary encoding of engineer and the remaining positions are *'s. If Q is issued to DB, and computed by comparing it against each entry of DB, it will return only those strings that match the specified field even though Q might actually represent an exponential number of strings. However, if Q is issued to NDB, it will be necessary to find which of all the possible strings of length l whose defined positions correspond to "engineer" are *not* in NDB and output them. It is an $\mathcal{NP}$-hard problem to accomplish this under this representation of NDB for an arbitrary choice of defined positions. However, as was discussed in Section 4.1.1 it is possible to construct NDB's with specific structures for which more complex queries can be answered efficiently. Intuitively, what makes some queries inefficient is not the size of NDB, as it is only polynomially larger than DB, but the fact that a single element of U, a single tuple, is represented by several NDB entries and that a single NDB entry represents several tuples. This makes it difficult to determine even if there are any engineers at all in DB.

In summary, under the current scheme, only authentication queries —queries that decide on the membership of a completely specified string—are supported efficiently; queries of an exploratory nature will, in general, be intractable. One of my future goals is to control this complexity boundary, either through a deeper understanding of the existing representations or by devising new ones. This would allow us to support a limited set of queries (say, those allowed by law) and prevent arbitrary exploratory searches.

4.3.2 Set Intersection

One potential use of negative databases is for privately computing the intersection of several sets. This primitive has applications as varied as privately matching sets of preferences in a dating website, finding entries common to a collection of watch lists or determining common transaction between two banks. Due to the inherent properties of negative databases it is possible to privately perform these computations in a very natural way.

Take for instance n parties, each an owner of some database DB_i, that wish to establish which items they have in common, i.e. $\{DB_1 \cap \cdots \cap DB_n\}$ without revealing the totality of the contents of their database or its cardinality. If each party produces a negative database NDB_i representing all records not in their DB_i to share with the other parties, then, noting that, by De Morgan's Law, $x \in \{DB_1 \cap \cdots \cap DB_n\} \iff x \notin \{NDB_1 \cup \cdots \cup NDB_n\}$, the i^{th} party can compute the set intersection by simply establishing which of the entries of its database DB_i are not in $\{NDB_1 \cup \cdots \cup NDB_n\}$, i.e. $DB_i - \{NDB_1 \cup \cdots \cup NDB_n\}$. This is an operation that can be carried out efficiently as discussed in Section 4.3.1.

This simple scheme would not only protect the identity of all entries outside the intersection, but also the cardinality of each party's private database as long as each NDB_i is hard to reverse (Sections 4.2, 4.4) or if the origin of each record in $\{NDB_1 \cup \cdots \cup NDB_n\}$ is concealed.

4.4 Negative Database Algorithms

The prefix algorithm presented in Section 4.1.1 is simple and demonstrates that a compact negative representation NDB can be obtained from DB. Although I have demonstrated in Section 4.2 that the general problem of reversing a given set NDB to obtain DB is $\mathcal{NP}$-hard, using the simple prefix algorithm to obtain NDB from DB raises two concerns regarding privacy: (a) The prefix algorithm produces only an easy subset of possible NDB instances, and (b) If the action of the prefix algorithm (or any algorithm) that produces NDB from DB could be reproduced by an adversary, then the adversary could easily decide for a given NDB and candidate DB whether NDB represents $U - DB$. The two concerns are of course related, for if an algorithm were capable of producing only one NDB for each DB it is given as input, the image of the algorithm could not define an $\mathcal{NP}$-hard set of instances of NESR.

In this section I present algorithms which address both of these concerns. The section is divided into two subsections, the first addresses how to create an initial negative database while the second deals with how it can be updated to reflect changes in the composition of DB. In addition, each subsection analyzes the algorithm's correctness and examines some of its properties.

4.4.1 Initialization

The $RNDB$ algorithm in Figure 4.4 takes as input a positive database DB (which might be initially empty) and outputs a negative database NDB (chosen uniformly at random from the set of possible NDBs) that exactly matches $U - DB$. Its basic strategy is, for a given permutation π—an ordering of the bit positions of a string—applied to every string in DB, to find every prefix V_p not present in $\pi(DB)$, augment each V_p with additional bit positions chosen at random (see Lemma 4.4.1.1 below) and randomly select from the resulting pattern a subpattern that subsumes it (see Definition 4.4.1.1 below).

Correctness

Definition 4.4.1.1. *Subsumption*: A string y is subsumed by string x if and only if every string matched by y is also matched by x. A string x obtained by replacing some of y's defined positions with don't cares, subsumes y.

Lemma 4.4.1.1. A set of 2^n distinct strings that are equal in all but n positions match exactly the same set of strings as a single string with those n positions set to the don't care symbol.

Lemma 4.4.1.2. Pattern_Generate(DB,V_{pe}) outputs a string that matches every string matched by the input pattern V_{pe} without matching any other strings in DB.

```
**Randomize_$NDB$($DB$,$l$)**
Let $w_i$ denote an $i$-bit prefix and $W_i$ a set
of $i$-length patterns.
0.   Find a random permutation $\pi$ and apply it to $DB$.
1.   Randomly select $1 \leq i \leq O(log_2(l))$
2.   Initialize $W_i$ to the set of every pattern of $i$ bits.
     Repeat
3.       Set $W_{i+1}$ to every pattern not present in
         $\pi(DB)$'s $w_{i+1}$ but with prefix in $W_i$
4.       for each pattern $V_p$ in $W_{i+1}$ {
5.         Randomly choose $1 \leq j \leq O(l)$
6.         for $k = 1$ to $j$ do {
7.           Randomly select an additional
             $0 \leq n \leq O(log_2(l))$ distinct positions.
8.           for every possible bit assignment $V_q$ of the
             selected positions (a total of $2^n$ patterns){
9.             $V_{pe} \leftarrow V_p \cdot V_q$
10.            $V_{pg} \leftarrow$Pattern_Generate($\pi(DB), V_{pe}$)
11.            Append $\pi'(V_{pg})$ to $NDB$.}}}[a]
12.      Increment $i$ by one
13.      Set $W_i$ to every pattern in $DB$'s $w_i$
14.  Until $i = l$ or $W_i$ is empty.
```

[a] π' denotes the inverse permutation of π.

Figure 4.4: The Randomize_NDB ($RNDB$) algorithm randomly generates a negative database representing the strings in $U - DB$.

Proof. To see that the algorithm (Figure 4.5) produces a string that matches everything V_{pe} matches, it suffices to note that the output string specifies a subset of the positions set in the input pattern V_{pe}: lines 1–6 discard some of the positions that comprise V_{pe}, while lines 7–9 reinstate some of them (see Definition 4.4.1.1).

Additionally, the subpattern found in lines 1–6 (a c-key according to Definition 4.4.1.2 from the following subsection) is guaranteed not to match any string in DB (lines 3–4). This subpattern is included in the final string output by the function,

```
Pattern_Generate($DB$, $V_{pe}$)
1.  Find a random permutation $\pi$.
    Let $n \leftarrow |V_{pe}|$
2.  for $i = 1$ to $n$ do {
3.      Construct a pattern $\pi(V_{pe})^{\dagger}$ with all but the $i^{th}$ bit from $\pi(V_{pe})$
4.      if $\pi(V_{pe})^{\dagger}$ not in $\pi(DB)${
5.          $\pi(V_{pe}) \leftarrow \pi(V_{pe})^{\dagger}$
6.          Keep track of the $i^{th}$ bit in a set indicator vector (SIV) }}
7.  Randomly choose $0 \leq t \leq |SIV|$
8.  $R \leftarrow t$ randomly selected bits from SIV
9.  Create a pattern $V_k$ using $\pi(V_{pe})$, the bits indicated by $R$ and *
    symbols in the remaining positions.
10. Return $\pi'(V_k)$. [a]
```

[a] π' is the inverse permutation of π.

Figure 4.5: Pattern_Generate produces a string over $\{0, 1, *\}$ that matches V_{pe} without matching any string in DB.

ensuring it will not match any string in DB.

□

Theorem 4.4.1.1. The Randomize_NDB algorithm, under any sequence of random choices, produces an NDB that exactly represents $U - DB$.

Proof. Let ns_j be any string in $U - DB$ and let i be the length of the smallest prefix V_p of ns_j that is absent from DB under permutation π. The algorithm will find this prefix at iteration i (line 3) and will insert a series of strings into NDB that match the same strings as V_p as follows: Lines 7–11 create a collection of strings that subsume V_p by augmenting it with additional positions (lines 7–9)(see Lemma 4.4.1.1) and assigning every possible pattern to these positions. Then, for each augmented pattern, function Pattern_Generate (line 10) creates a string that subsumes it without matching anything in DB (see Lemma 4.4.1.2). The resulting string is finally inserted into NDB (line 11).

In the case where DB is empty, lines 1–3 will consider the strings represented by every possible pattern of length $i+1$ in the $i+1$ length prefix (under permutation π), which encompasses all of U. Lines 4–11 insert the appropriate strings into NDB as discussed above. The function iterates once and exits.

□

Properties

Section 4.2 presents a transformation from 3-SAT to NDB, in what follows, I will use the formalisms interchangeably. In particular, DB and sets of assignments will be used interchangeably, NDB and formula ϕ will be used interchangeably, and the output of the algorithms to be presented in this section can be viewed either as strings in NDB or clauses in ϕ. For this reason clauses in ϕ are restricted to have no repeated variables.

The algorithm presented in Section 4.4.1 has the flexibility to produce NDBs or SAT formulae with varying structures that represent the same set of binary strings. The ease with which a formula can be solved (or an NDB reversed) depends on this structure (see References [100, 32, 34]). The following are some properties of the outputs the discussed algorithm is able to produce.

Definition 4.4.1.2. *c-key*: A c-key is bit pattern [3] not present in DB with no extraneous bits, i.e. a c-key defines a minimal pattern in $U - DB$ in the sense that the removal of any bit yields a pattern in DB (see Figure 4.2). A $\bar{c}$-key is the bitwise complement of a c-key.

Lemma 4.4.1.3. Let DB be a set of assignments and ϕ a CNF formula. ϕ is

[3] Note that the bits in the pattern need not occupy contiguous positions in the string.

satisfied by every $x \in DB$ if and only if every clause C_q in ϕ contains a $\bar{c}$-key with respect to DB.

Proof. Suppose clause C_q of ϕ contains a $\bar{c}$-key. Then, by Definition 4.4.1.2, no $x \in DB$ exhibits the corresponding c-key. Each $x \in DB$ contains at least one bit appearing in $\bar{c}$-key which satisfies the corresponding literal and therefore satisfies C_q.

Now assume each $x \in DB$ satisfies each clause of ϕ (that is, each x is a satisfying truth assignment for ϕ). Suppose to the contrary, that some clause C_q does not contain a $\bar{c}$-key. Then, the complement pattern of $\bar{c}$-key appears in DB, and in particular in at least one $x \in DB$. But then x contains no bit appearing in $\bar{c}$-key, thus failing to satisfy each of the corresponding literals in C_q. This contradicts our original supposition, hence, it must be that every clause C_q contains a $\bar{c}$-key.

□

Lemma 4.4.1.4. For every possible clause satisfied by DB contained in the input pattern V_{pe}, there is some execution of Pattern_Generate (Figure 4.5) (with an appropriate sequence of random choices) that will generate it.

Proof. Let C_q be a clause satisfied by DB and P_q its corresponding bit pattern. Suppose P_q is contained in the input pattern V_{pe}, then by Lemma 4.4.1.3 it must have as a subpattern some c-key K. For every pattern V_{pe} and every c-key K contained in V_{pe}, there exists a permutation π such that K occupies the $|K|$ rightmost bit positions of $\pi(V_{pe})$ (step 1). The algorithm proceeds by discarding one by one, from left to right, every bit it examines for as long as there is a c-key present within the remaining subpattern (steps 2–6). It follows that since K is a c-key and occupies the $|K|$ rightmost positions of $\pi(V_{pe})$ that K is the pattern that will be found[4]. Steps

[4] Note that it is not required for the c-key to be contiguous or to occupy the rightmost bits to be found. It is only convenient to focus on this case for the proof.

7–9 of the algorithm generate a string containing K plus, by the appropriate random choice, the additional bits than comprise C_q.

□

Lemma 4.4.1.5. For every clause satisfied by DB there is at least one string in $U - DB$ that contains the corresponding pattern.

Proof. Suppose C_q is a clause satisfied by DB and P_q the corresponding bit pattern, then by Lemma 4.4.1.3 C_q has a $\bar{c}$-key and P_q a c-key K. By the definition of c-key (Definition 4.4.1.2) there is no string in DB with K as a subpattern, hence every string with K as a subpattern must be in $U - DB$, including the one containing P_q. □

Theorem 4.4.1.2. The $RNDB$ algorithm, during any execution, can produce any clause with $O(log_2(l))$ or fewer literals that is satisfied by DB.

Proof. Let C_q be a clause of $k \leq O(log_2(l))$ literals satisfied by DB and P_q its corresponding bit pattern. For each P_q there is at least one string N_c in $U - DB$ that contains it (Lemma 4.4.1.5). String N_c, under permutation π, has a prefix of length i that is not present in DB which will come under consideration at iteration i of the algorithm (line 3). Suppose m of the k bits of P_q are included in the i length prefix of N_c, the remaining $k-m$ positions will be set in steps 7–8 by the appropriate random choice and the string corresponding to C_q will be found by Pattern_Generate (Lemma 4.4.1.4).

The cycle of line 5 ensures that each prefix is considered $O(l)$ times, allowing any particular clause contained within a string with that prefix to be found independently.

□

Corollary 4.4.1.1. The $RNDB$ algorithm can produce any sequence of $O(l)$ clauses with $O(log_2(l))$ literals that are satisfied by DB as part of its output.

Proof. Theorem 4.4.1.2 states that any clause satisfied by DB, can be generated during any execution of the algorithm. It follows that, since the algorithm can generate formulas with $O(l)$ clauses, it can generate any sequence of $O(l)$ clauses that are satisfied by DB as part of its output.

□

It is important to note that the $RNDB$ algorithm is unable to produce every (polynomial size) formula (in polynomial time) that is satisfied exactly by DB. In fact, it can be shown that there is no efficient algorithm that, given DB as input, can generate all and only formulae that are exactly satisfied by DB, unless $Co\mathcal{NP} = \mathcal{NP}$. We saw, however, that the algorithm can generate every formula of a given length that is satisfied exactly by DB together with clauses that are superfluous[5] (Corollary 4.4.1.1).

It was shown in [54] (see Appendix B) that the image of $RNDB$ algorithm does in fact define an $\mathcal{NP}$-hard problem as a function of the size of the resulting NDB albeit not necessarily as a function of the size of the original DB. Further, given that $\mathcal{NP}$-hardness is a worst case analysis, this property alone is usually not sufficient for a negative database to be hard to reverse in practice. My aim is ultimately to generate instances that are hard to reverse on average.

Finally note that Pattern_Generate runs in time $O(l \cdot |DB|)$ and that the Randomize $\_NDB$ algorithm outputs a database with $O(l^2|DB|)$ entries in $O(l^3|DB|^2)$ time.

[5] Implied by this observation is that identifying superfluous clauses is an $\mathcal{NP}$-hard problem.

Randomize_NDB({},4)	**Delete**(1111,NDB)	**Insert**(1111,NDB)
000*	000*	000*
001*	001*	001*
01*0	01*0	01*0
01*1	01*1	01*1
10*0	10*0	10*0
10*1	10*1	10*1
111*	110*	110*
110*	11*0	11*0
	*110	*110
		11

Figure 4.6: Possible states of NDB after successive initialization, deletion and insertion of a string.

4.4.2 Updates

Databases are sometimes used as archives for preserving information and need not afford the deletion of records or even the insertion of new ones. However, a vast number of applications do perceive the database as a dynamic entity capable of being updated. The previous sections have shown how a negative database can be created to store or represent $U - DB$, in this section I present three basic operations that allow modifying it once it has been initialized: Insert, Delete and Clean-up. It is worth mentioning that the meanings of the insert and delete operations are inverted from their traditional sense, since we are storing a representation of what is *not* is some database DB. For instance, the operation "insert x into DB" is implemented as "delete x from $U - DB$" and "delete x from DB" as "insert x into $U - DB$" (see Figure 4.6).

The core operation for the procedures, named Negative_Pattern_Generate (Figure 4.7), creates a string over $\{0, 1, *\}^l$ that subsumes x and matches nothing else in DB. Its functionality is similar to that of Pattern_Generate (Figure 4.5) and could be

```
Negative_Pattern_Generate($NDB$, $x$)
1.  Create a random permutation $\pi$
2.  for all specified bits $b_i$ in $\pi(x)$
3.      Let $x'$ be the same as $\pi(x)$ but with $b_i$ complemented
4.      if $x'$ is subsumed [a] by some string in $\pi(NDB)$
5.          Keep track of the $i^{th}$ bit value in a set indicator vector (SIV)
6.          Set the value of the $i^{th}$ bit of $\pi(x)$ to the * symbol
7.  Randomly choose $0 \leq t \leq |SIV|$
8.  $R \leftarrow t$ randomly selected bits from SIV
9.  Create a pattern $V_k$ using $\pi(x)$ and the bits indicated by $R$.
10. return $\pi'(V_k)$ [b]
```

[a] See Definition 4.4.1.1 in Section 4.4.1.

[b] π' is the inverse permutation of π.

Figure 4.7: Negative_Pattern_Generate. Takes as input a string x defined over $\{0, 1, *\}$ and a database NDB and outputs a string that matches x and nothing else outside of NDB.

replaced by it. However, the difference is that Negative_Pattern_Generate does not need DB to be available, a potentially useful feature for applications in which keeping a copy of DB is a liability. This variation is reflected in lines 3–5 where extracting a subpattern [6] from input x is accomplished by determining if replacing a specified bit in x by a don't care symbol yields a string that is represented by $NDB \cup \{x\}$. Owing to the similarity between procedures the proof that Negative_Pattern_Generate is correct is very similar to Lemma 4.4.1.2 and is therefore omitted.

Insert into NDB

Consider a database that records the member names of some exclusive club, suppose one of the constituents has failed to meet his obligations and his membership is

[6] Note that this subpattern does not necessarily constitute a c-key (it is easy to see that extracting c-keys from NDB is $\mathcal{NP}$-hard).

```
Insert($x$, $NDB$)
Randomly choose $1 \leq j \leq O(l)$
for $k = 1$ to $j$ do
    Randomly select $0 \leq n \leq O(log_2(l))$
    Randomly select from $x$, $n$ distinct unspecified bit positions
    for every possible bit assignment $V_p$ of the selected positions
        $x' \leftarrow x \cdot V_p$
        $y \leftarrow$ Negative_Pattern_Generate($NDB$, $x'$) [a]
        add $y$ to $NDB$
```

[a] Note that Pattern_Generate(DB, x') can be used instead, provided DB is available.

Figure 4.8: Insert into NDB.

revoked. Removing his name from the list will accomplish this task. However, if a negative database is used as a surrogate, his name must be added to the archive of all character combinations which are not member names. In this section I present an operation that accomplishes this task for any string $x \in \{0, 1, *\}$. The algorithm has the flexibility of simply appending x to NDB, tacitly leaving a trace that x was removed from DB, or to more subtly incorporate it by performing some additional operations. Figure 4.8 outlines this procedure.

Theorem 4.4.2.1. Function Insert(x, NDB) outputs a negative database that exactly matches $(U - DB) \cup \{x\}$.

Proof. It follows directly from Lemma 4.4.1.1 and Lemma 4.4.1.2.

□

Delete from NDB

Consider having a database of the member names of some club, and wanting to include a new recruit. Adding the name to the list will suffice for a positive database; for a negative database, on the other hand, the operation involves removing any reference to the name from NDB. In this section I present an operation that accomplishes this task. Note that the object of the operation is to stop a binary string from being represented in NDB, i.e. prevent it from being matched by any entry in NDB, rather than eliminating a specific NDB record. Also, bear in mind that only the strings specified for deletion should cease from being represented. Figure 4.9 gives the pseudocode for the algorithm.

Theorem 4.4.2.2. Delete(x, NDB) outputs a negative database that exactly matches $U - (DB \cup \{x\})$.

Proof. Lines 1–2 identify the subset, D_x, of NDB that matches x and removes it from NDB. Note that there is no string in $NDB - D_x$ that matches any binary string matched by x.

Lines 3–7 reinsert all the strings represented by D_x except x: For each string y in D_x and for each of its unspecified positions (don't care symbols) there is a string y_i created which differs from x in its i^{th} position (line 6) and inserted into NDB (see Theorem 4.4.2.1). None of the new strings y_i match x.

If a string $z \in \{0,1\}^l$ other than x is matched by some $y \in D_x$ then z must have the same specified positions as y. Given that z is different from x it follows that it must disagree with it in at least one bit, say bit k; z will be matched by y'_k. Therefore only x is eliminated from NDB. Finally, observe that since y subsumes each new entry y'_i (see Definition 4.4.1.1) no unwanted strings are included by the operation. □

```
Delete($x$, $NDB$)
Let $D_x$ be all the strings in $NDB$ that match $x$
Remove $D_x$ from $NDB$.
for all $y \in D_x$
    for each unspecified position $q_i$ of $y$
        if the $i^{th}$ bit of $x$ is specified
            Create a new string $y_i$ using the specified bits of $y$ and the
            complement of the $i^{th}$ bit of $x$.
            Insert($y_i$, $NDB$)
```

Figure 4.9: Delete from NDB.

One important effect of the Insert and Delete operations is that they both cause NDB to grow, especially in the latter case when the number of new entries in NDB is a function of the number of entries matched by the strings to be deleted. To address this problem I introduce a clean-up operation designed to reduce the size of the negative database and thus reduce the number of entries expected to match any binary string.

Clean-up

The operation presented here (Figure 4.10) takes as input a negative database NDB and outputs a negative database NDB' that represents exactly the same set of binary strings, and therefore, matches exactly those strings not in DB. The function includes a parameter τ (line 4) which is meant to drive the size of the resulting database. If the Insert operation introduces fewer than τ entries per call then Clean-up will not increase the size of NDB and will likely reduce it.

Theorem 4.4.2.3. Clean-up outputs a negative database that represents the same set of binary strings as its input NDB.

```
Clean-up($NDB$, $\tau$)
1. Randomly select a string $x$ from $NDB$
2.     Find a subpattern $K$ of $x$ not found in any $DB$ string [a]
3.     Let $D_K$ be all strings in $NDB$ that have $K$
4.     if $|D_K| > \tau$
5.         Remove $D_K$ from $NDB$
6          Create a string $V_K$ of length $l$ with $K$ as
           a subpattern and the remaining positions set to *
7.         Insert($V_K$, $NDB$)
```

[a] According to lines 1–6 of Fig.4.7 or lines 1–6 of Fig. 4.5.

Figure 4.10: Clean-up. Outputs a negative database that represents the same strings as its input NDB with equal or fewer entries.

Proof. Lines 1–2 find a subpattern K of a string in NDB, such that no string in DB has that pattern (see Definition 4.4.1.2 Lemma 4.4.1.2), i.e. every string in $\{0,1\}^l$ with such a pattern must be represented in NDB. Line 3 finds all NDB entries D_K that exhibit this pattern, line 5 removes them. Only strings in $\{0,1\}^l$ that have K stop being represented in NDB, for if a string y is matched by D_K then it must also be matched by K. Therefore, the removal of D_K causes only strings with K as a subpattern to be excluded. Line 6–7 reinsert every string and only strings with K as a subpattern into NDB (see Theorem 4.4.2). □

It is interesting to note that the Clean-up operation can be used to transform a negative database into another representing exactly the same set of binary strings (see Reference [50] for some experimental results). This feature may be useful in a security setting in which we wish to hinder the ability of comparing databases, and as means for transforming a database into another that is easier or harder to reverse for a given heuristic.

Properties

It was previously mentioned that Pattern_Generate could be used in place of Negative _Pattern_Generate within the Insert and Delete operations. In the case of Clean-up, extraction of a minimal pattern (line 2) can be achieved with lines 1–6 of Pattern_Generate or Negative_Pattern_Generate depending on the availability of DB. If the former is used, it is easy to see that the resulting negative database preserves the properties of the $RNDB$ algorithm's output outlined in Section 4.4.1. On the other hand, if the latter is applied then it is not feasible to determine if a pattern constitutes a c-key, and therefore, the number of clauses that can possibly be generated will be restricted.

An important property of the Insert, Delete and Clean-up operations is that, in general, their application does not make the problem of reversing a given NDB any easier. Consider the following problem:

Definition 4.4.2.1. Self-Recognition-Pair (SR-Pair)

INSTANCE: (ϕ, S, ϕ', S') where ϕ is a SAT instance, S a set of assignments to ϕ, ϕ' is a SAT instance obtained by inserting or deleting an arbitrary assignment x and only x from ϕ by means of any polynomial time algorithm $\mathcal{A}$. S' is obtained by inserting or deleting x from S accordingly.

QUESTION: Is ϕ' exactly satisfied by S'?

Theorem 4.4.2.4. SR-Pair is $\mathcal{NP}$-hard

Proof. We prove the theorem by reducing SAT to SR-pair. The proof is divided into the case in which $\mathcal{A}$ is used to insert a satisfying assignment x to ϕ and the case in which it is used to delete a satisfying assignment x from ϕ.

1. Insertion version of SR-Pair is $\mathcal{NP}$-hard.

Given instance ϕ of SAT. Pick any assignment x. If x satisfies ϕ answer YES to instance ϕ of SAT. If x does not satisfy ϕ use $\mathcal{A}$ to create ϕ' that is exactly satisfied by the assignments which satisfy ϕ, union $\{x\}$. Then (ϕ, $\{\}$, ϕ', $\{x\}$) is a valid instance of the insertion version of SR-Pair and:

ϕ is a NO instance of SAT $\iff$ (ϕ, $\{\}$, ϕ', $\{x\}$) is a YES instance of SR-Pair.

2. Deletion version of SR-Pair is $\mathcal{NP}$-hard.

   Given an instance ϕ of SAT. Pick any assignment x. If x satisfies ϕ answer YES to instance ϕ of SAT. Otherwise, x does not satisfy ϕ and use $\mathcal{A}$ to create ϕ' that is exactly satisfied by the assignments which satisfy ϕ, minus $\{x\}$ (note ϕ is logically equivalent to ϕ'.) Then (ϕ, $\{\}$, ϕ', $\{\}$) is a valid instance of the deletion version of SR-Pair and:

   ϕ is NO instance of SAT $\iff$ (ϕ, $\{\}$, ϕ', $\{\}$) is a YES instance of SR-Pair.

   We conclude by stating that there is a polynomial time reduction from SAT to SR-Pair and hence that SR-Pair is $\mathcal{NP}$-hard.

□

It follows that the application of the Insert, Delete and Clean-up operations doesn't make a difficult instance any easier to reverse. However, let me stress that the practical reversal difficulty of a specific NDB depends on the heuristics used to solve it, and hence these operations can decrease or increase the actual time required by a given heuristic.

The complexity of the algorithms can be broken down as follows: Negative _Pattern _Generate runs in time $O(l \cdot |NDB|)$. Insert takes $O(l^3|NDB|)$ time if Negative _Pattern _Generate is used, or $O(l^3|DB|)$ if Pattern_Generate is employed and inserts $O(l^2)$ strings per call into NDB. The Delete operation runs in $O(l^4|NDB|^2)$ or $O(l^4|NDB||DB|)$ time depending on whether the negative or positive pattern generate procedures are used. Delete causes the addition of $O(l^2|NDB|)$ entries in NDB. The Clean-up time complexity is dominated by its call to Insert and has the same complexity. Note that these bounds are due, in great part, to the generality that the algorithms afford. It is expected that the production of hard NDB instances will require limiting some parameters which will, in turn, reduce the complexity of the operations.

4.5 Summary

In this chapter I introduced the concept of negative representations of information and presented a specific instantiation of this idea called negative databases. I established that a negative database can be constructed in time polynomial in the size of its positive counterpart. I presented algorithms for creating and maintaining such a database and offered an analysis of their properties and the properties of the negative databases they produce. Further, I investigated one characteristic of negative databases, namely that given a negative database it is an $\mathcal{NP}$-hard problem to recover its positive image. I also showed that, even though reversing a negative database is hard, there are certain types of queries that can be carried out efficiently, and discussed how this property can be exploited to privately compute the intersection of two sets. I am optimistic that, by tailoring a negative representation to particular requirements, the scheme can address at least some of the problems presented by large collections of sensitive data.

Chapter 5

Partial Negative Databases

A database DB can be conceptualized as a set of strings or records that naturally partitions the universe of possible items or strings U into two subsets—those stored in the database and those left outside of it. When a database holds the items we care about it is referred to as a *positive database*; it is referred to as a *negative database* when it represents all but those records. The previous two chapters have been concerned with the possibility of creating negative databases; Chapter 3 shows how to negatively represent all of the strings not in the crossover closure of some input set S, and Chapter 4 discusses how to efficiently create an exact negative image of any set of strings. I discussed some of the characteristics of these databases and their encodings, and demonstrated how to perform basic operations on them.

In this chapter I investigate an additional property of describing a database by its complement, one concerned with the amount of information each string in DB and $U - DB$ has, and with how queries are answered using either set. Both DB and $U - DB$ contain the same amount of information—both sets can answer the same questions— however, the manner in which information is distributed among the individual entries and the way in which query answers are inferred from each database are aspects that are fundamentally distinct between them; differences that might lead to a more natural implementation of some current applications and to the facilitation of new ones.

5.1 Information Content

In this section I explore how information is distributed among the strings of DB and $U - DB$. It is worth clarifying that what is meant by information here does not refer to the content of the specific records but rather to the state of our knowledge regarding the partition of U induced by DB. I am concerned with how it changes as the membership of strings in U is revealed.

Let U be the set of all strings of length l defined over the binary alphabet $\{0, 1\}$. For the current analysis I assume that the size of DB is known[1], and that records in DB are i.i.d., i.e. there is nothing in any string which suggests the membership of any other string in particular. In general, I assume that the size of DB is considerably smaller than that of its complement, that is, $|DB| \ll |U - DB|$.

Let:

DB be the positive database,

DB_f be a subset of DB,

UDB denote $U - DB$, and

UDB_f be a subset of $U - DB$.

The probability of an arbitrary string x belonging to DB when only a fraction of the positive or negative database is available for inspection can be written as:

[1] Not knowing the size of DB a-priori introduces additional uncertainty.

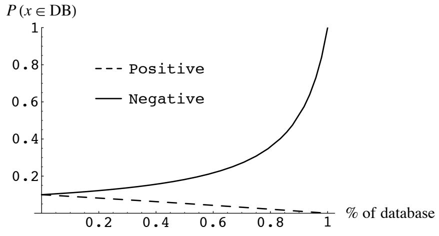


Figure 5.1: Probability of an arbitrary string x belonging to DB when only a fraction of the positive or negative database is available. The plot illustrates the probabilities when $|DB|$ is 10% of $|U|$.

$$P(x \in DB | x \notin DB_f) = \frac{|DB - DB_f|}{|U - DB_f|} \tag{5.1}$$

$$P(x \in DB | x \notin UDB_f) = \frac{|DB|}{|U - UDB_f|} \tag{5.2}$$

Likewise the probability that a given string x is a member of UDB:

$$P(x \in UDB | x \notin DB_f) = 1 - P(x \in DB | x \notin DB_f) \tag{5.3}$$

$$P(x \in UDB | x \notin UDB_f) = 1 - P(x \in DB | x \notin UDB_f) \tag{5.4}$$

Figure 5.1 shows how the probability of a certain record x belonging to DB changes as strings in the database are inspected (without encountering x). The graph illustrates one important and potentially useful feature of using the negative image of a set, namely that the probability of an arbitrary string x belonging to DB increases as the identity of strings in UDB becomes known whereas it decreases as positive records become available.

Using the above results (eq. 5.1–5.4) I now turn to the average amount of information or entropy that is associated with a given partition of U and discuss how it is affected as the membership of strings is revealed. For the purpose of the current discussion I consider the entropy, H_P, when a fraction of DB is known, and the entropy, H_N, when a fraction of UDB has been disclosed. The mixture case can be easily extrapolated from them.

$$\begin{aligned} H_P &= -P(x \in DB | x \notin DB_f) \cdot \log_2 \left(P(x \in DB | x \notin DB_f) \right) \\ &- P(x \in UDB | x \notin DB_f) \cdot \log_2 \left(P(x \in UDB | x \notin DB_f) \right) \qquad (5.5) \\ H_N &= -P(x \in DB | x \notin UDB_f) \cdot \log_2 \left(P(x \in DB | x \notin UDB_f) \right) \\ &- P(x \in UDB | x \notin UDB_f) \cdot \log_2 \left(P(x \in UDB | x \notin UDB_f) \right) \qquad (5.6) \end{aligned}$$

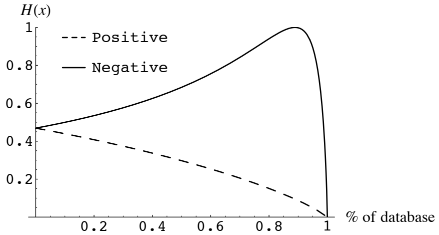


Figure 5.2: The uncertainty associated with U once the membership of a subset of DB or $U - DB$ is known. The plot illustrates the entropy of the conditional distributions starting with a $|DB|$ that is 10% of $|U|$.

Figure 5.2 shows how the entropy changes as strings in one of the sets are disclosed for the case in which DB is 10% of U. It is interesting to see how the entropy of the conditional distributions increases as the identity of strings in the negative image becomes known (up to the point when the size of the unknown negative is equal to the size of the unknown positive), and how it always decreases as the members of DB are revealed without encountering x. Note that this feature is due to the asymmetry in the size of the two sets and one that can be useful once the handling of either set becomes a feasible option.

5.1.1 Querying DB and $U - DB$

In the previous section we saw that the entropy of the conditional distribution regarding the membership of an arbitrary string in $U - UDB_f$ increases as UDB_f grows (up to a certain point, see Figure 5.2). It is important to point out, however, that the uncertainty concerning the identity of strings in DB decreases (their self-information tapers off), ensuring that questions about DB can be answered by inspecting UDB.

This leads to an important distinction between storing positive or negative representations of data. I have argued that both DB and UDB can answer the same questions but there is a difference in how an answer is inferred using either of the sets. Consider the following:

Using UDB

$$\begin{aligned} x \in DB &\iff \forall_{y_i \in UDB} \quad x \neq y_i \\ x \notin DB &\iff \exists_{y_i \in UDB} \quad x = y_i \end{aligned}$$

Using DB

$$\begin{aligned} x \in DB &\iff \exists_{y_i \in DB} \quad x = y_i \\ x \notin DB &\iff \forall_{y_i \in DB} \quad x \neq y_i \end{aligned}$$

To ascertain the membership of a string x using UDB all of UDB must be searched if x belongs to DB, whereas the search can halt as soon as x is found if x is not in DB. Conversely, if DB is being used to answer questions the opposite is true: if x is in DB the search can stop when x is found, and all DB must be searched otherwise. This is not to say that questions about DB cannot be answered using UDB_f but rather that answers should be qualified to reflect their certainty.

To illustrate this point consider the case where either half of DB or half of UDB are available for querying. Suppose we wish to determine if an arbitrary string x belongs to DB, and we may consult only half of UDB; we will be able to answer with absolute certainty that x is not in DB if it is present in the half of UDB which we hold. On the other hand, if x is not in our share (which will happen for $\frac{|UDB|}{2} + |DB|$ strings) we will be forced to qualify the answer with the probability that x belongs to DB given that it is absent from our share (see eq. 5.2). The situation is inverted when only half of DB is available; here, we can be sure that x is in DB if x is in the half of DB at our disposal, and we would need to qualify the answer for all other cases.

This is significant in a scenario where each item in DB is in itself valuable. Suppose DB is a list of passwords; one way to protect their identity is to split UDB (all strings except the passwords in DB) into two halves. The owner of each half cannot be certain of the identity of any valid password, without consulting the other half. Splitting DB gives each party half of the valid passwords, any of which is sufficient to gain access to the protected resource.

5.2 Negative Databases

The previous sections dealt with how information is distributed among binary strings in DB and in $U-DB$, and how answers may be inferred using either set. However, it is generally infeasible to explicitly store the strings of $U-DB$—recall the assumption that $DB \ll U-DB$. Chapter 4 explores the concept of negative databases whereby the set $U-DB$ is compactly represented by a set NDB whose strings are defined over $\{0,1,*\}^l$ where l is the original string length. One consequence of this representation is that a single NDB entry typically represents several strings in $U-DB$ and that, for some classes of negative databases, a member of $U-DB$ may be represented by several NDB entries [2].

I now turn attention to estimating the amount of information each string in NDB possesses. The following calculations are simplified by assuming that all negative records are independent of one another, that they are generated uniformly at random with replacement, and that the size of NDB is small relative to the number of possible NDB stings. In practice, the records in the negative database are not independent, and are not generated in this fashion as it is important for none of them to match any string in DB, and for all strings in $U-DB$ to be accounted for. However, these assumptions are reasonable when DB is small and NDB is generated by a suitable randomized algorithm, such as the one described in Chapter 4.

Let $p_i = 2^{-r_i}$ be the proportion of strings in U matched by an individual NDB record with r_i specified bits (see Section 4.1). The number of strings in U matched by n negative records can be written as:

$$|U-DB|(1-\prod_{i=1}^{l}(1-p_i)^{n_i}) \tag{5.7}$$

where $n=\sum n_i$ and n_i is the number of strings with r_i specified bits.

To simplify matters further and gain insight into the effects of the representation on how information is dispersed in a negative database, the remainder of the analysis will be carried out for a specific class of NDBs in which each string has exactly r specified positions. The algorithms in Chapter 4 and in Reference [50] can readily

[2] This is not the case of NDB's generated by the prefix algorithm (see Section 4.1.1).

generate databases with this structure. The number of strings in $U - DB$ matched by n such records is:

$$|U - DB|(1 - (1 - 2^{-r})^n) \tag{5.8}$$

Figure 5.3 shows how the set $U - DB$ is covered (strings matched) as more and more negative records are considered. Note the decrease in slope due to the expected number of matching overlaps among strings in NDB.

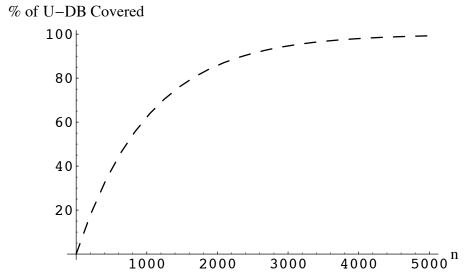


Figure 5.3: Percentage of $U - DB$ matched as a function of the number n of negative records for $r = 10$ specified positions.

Using the results of Section 5.1, it is straightforward to estimate the probability of a string x belonging to DB as a function of the number of inspected NDB records that do not match it. Equation 5.9 details this probability and Figure 5.4 shows a plot for a specific NDB.

$$P(x \in DB | x \notin NDB_f) = \frac{|DB|}{|U| - |U - DB|(1 - (1 - 2^{-r})^n)} \tag{5.9}$$

where NDB_f is a fraction of NDB and $|NDB_f| = n$.

Note that both Figures 5.4 and 5.3 exhibit asymptotic behavior that is an artifact of the assumptions used to estimate the number of strings matched by a set of

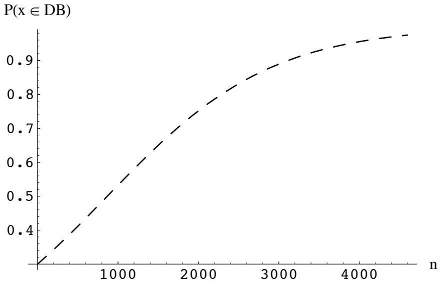


Figure 5.4: Probability that a string x belongs to DB given that it is not matched by n NDB strings with $r = 10$ specified positions.

negative records. In actuality, having all of NDB covers $U - DB$ exactly and can answer all queries regarding U.

5.3 Scenarios

In this section I outline two scenarios and operations that exploit and elucidate the differences of using a positive or a negative representation of data.

5.3.1 Distributed Negative Databases

In Chapter 4, I describe how negative databases can be used to perform set intersection. Recall that, by De Morgan's Law, a string x belongs in the intersection of a group of databases if and only if x is not in any of the corresponding negative databases. The security of the scheme discussed in that chapter relied on the assumption that each NDB was hard to reverse. I now relax this supposition and look at how the set intersection insight can be applied to a single database. Consider a negative database NDB whose records are distributed among n subsets $NDB_1 \ldots NDB_n$. Each NDB_i matches only a fraction of $U - DB$ (see Section 5.2), and is the negative image of some set $GDB_i = DB \cup G_i$, where G_i are those strings in $U - DB$ not matched by NDB_i, and act as noise that obfuscates the composition of DB. The union of the NDB_i yields the negative image of DB; the intersection of the GDB_is produces DB.

In the previous sections I studied how much information is embodied in an arbitrary subset of NDB—NDB_i in this case—and established that only some questions regarding items that are *not* members of DB can be answered with absolute certainty. Imagine a scenario in which each NDB_i is assigned to an agent A_i and that agents are independent of each other, able to consult only the database they manage. Some other entity J wishes to ascertain whether item x is in DB and is allowed to pose a query to each A_i. By the set intersection property, J can determine the membership of x by consulting the individual agents. Note however, that if it is the case that x is a member of DB all agents need be queried to confirm it as a fact. This brings out an important characteristic of the setup: Only J can establish with certainty that x is in DB and J has to consult all A_i's to do so.

Suppose a list of names has been drawn as a result of a lottery. The agency that holds the event wishes to communicate to the winners their good fortune. However, they want to avoid publishing the cleartext of the list to avoid impersonators, thieves, and to protect the privacy of the lucky ones. After reading this dissertation the board of directors has decided to create a negative database of the list, and distribute it among n parties. Each party may provide answers regarding their share but must disallow any single entity from making several consultations (allowing too many queries opens the door for a dictionary attack [3]). Also, they must protect their NDB_i from getting stolen.

Any person interested in learning whether they've won will need to consult the

[3] A dictionary attack simply uses a big list of names, such as the phone directory, and uses each one the query the database.

n agents, if their name is not in any of the negative databases, they may go to the benevolent agency and ask for their winnings. A variant of this might ask the holders of the partial databases to issue a certificate: "Name Y is not on NDB_i", the possession of the n certificates with your name, together with an Id, allows you to claim the prize. The benefit of the setup is that only the winners (and the agency) can be absolutely sure that they've won. The administrator of each NDB_i knows the name of some of the people that did not win, but cannot be certain of the name of anyone who actually did (see Section 5.1.1).

The security of this scheme relies on the fact that the negative database is distributed, each agent having a limited amount of information (see Section 5.2), rather than on having a negative database that is hard to reverse. Also, the distributed strategy allows for more complicated queries than simple membership; say the positive database DB has the tuples <name, profession, address> and the negative database NDB all possible tuples not in DB. If NDB is divided into n subsets, a query such as "What are the names and addresses of the engineers contained in DB" can be answered by issuing "Give me all the tuples <name, address> of the records having engineer in the profession field" to each NDB_i and compiling a new NDB' out of all the replies. Reversing NDB' will yield the desired answer. Note that NDB' has all the tuples <name, address> of all the engineers that are not in DB, complementing it (and I'm assuming NDB' is easy to reverse) will leave those that are.

This setup can be used in concert with other encryption techniques to yield some interesting applications, for instance: Some agency wants to communicate a message to a group of people. They want the message to be secret, the identity of the recipients to remain ambiguous, and to conceal which messages have been collected. One option is to encrypt the message using the name of the recipient as key, producing E(message), and create a database DB with the tuples <H(name), E(message)>

where H(name) is the hash of the name with a public one-way function[4], a scheme such as this is described in [60, 59]. Now instead of publishing DB, which is suceptible to a dictionary attack and lets the entity in charge of managing it know which messages have been retrieved, we can use the construct described above and create a negative database NDB and divide it amongst n parties. To see if you have a message simply hash your name to create H(name) and ask each NDB_i for all records with H(name) in their name field (concealing your identity). Inverting the union of the tuples $<E$(message)$>$ reported by all NDB_is will result in a database with your messages. Only you know which records it contains, only you know if its empty or not. If its not, the messages may be decrypted using your name.

A further characteristic of distributing a negative database worth pointing out is that inserting an item x into the definition of DB, i.e. deleting it from NDB, is a more difficult task than removing it, since all NDB_is have to be inspected and perhaps modified to assure x is subtracted from all of them (recall that there is a lot of redundancy in a NDB and x may be present in several NDB_is). Removing x from DB, on the other hand, is simple as it will suffice to append it to any NDB_i (see Section 4.4.2 of Chapter 4). Contrast this to performing the same operations on a distributed positive database; inserting x into DB is accomplished by inserting it into any subset, while removing it will entail making sure it is absent from all of its subsets. Resilience to deletions is a desirable property for data integrity. However, there are settings where it is useful to easily remove an item, such as when the database stores a checklist or when it keeps perishable messages. Resilience to insertion is advantageous in scenarios like the one described herein where you wish to prevent people from including their names in the list of winners.

Similar strategies can be devised to perform secret sharing [118, 17] that exhibit some of the nuances of using negative databases. Beyond the scenarios presented

[4] This can also be accomplished using a public key (but not everybody has one).

so far there are other possible uses of negative data representations. The following section presents a way in which it can be used to collect data privately.

5.3.2 Preference Ranking

Suppose you are given the task to determine which are the most prevalent sexual transmitted diseases at your university. Say you are given a list with ten diseases, and you are asked to rank them in the order of prevalence at your school. Imagine entering a classroom with the questionnaires in hand and asking students to please mark which of the listed diseases they have; they might feel their privacy is being threatened. Some will probably refuse to comply and others will likely provide inaccurate answers. Consider now a slightly different scenario; you again enter the room and again distribute the questionnaires, but now you ask the students to please, out of the ten diseases listed, put a mark next to one disease they have *not* had (in general, there will be many diseases any single person has not had).

The amount of information surrendered in a regular multiple-option positive questionnaire in which one option is to be selected as true is:

$$-\sum_i p_i \log_n p_i \tag{5.10}$$

where p_i is the probability that option X_i is true and n is the number of categories in the questionnaire. Here the maximum amount is obtained when all options are equally likely.

The quantity surrendered if the questionnaire is answered negatively can be obtained by subtracting, from (eq. 5.10), the amount information gained by the same questionnaire given that one option has been eliminated, and can be written as:

$$-\sum_{i} p_i \log_n p_i + \sum_{i \neq s} P(p_i | X_s = F) \log_n P(p_i | X_s = F) \tag{5.11}$$

where p_i is the probability that option i is true, n the number of categories, X_s the option that is negatively selected and $P(p_i|X_s = F)$ the posterior probabilities of each X_i being true after X_s has been selected as false.

Clearly, the information gained using the negative version of the questionnaire is bounded by the amount procured from its positive counterpart. One consequence of this, is that it is more mindful of the individual's privacy, as it does not require the subject to reveal a potentially embarrassing fact (such as one of the STDs that afflicts him). It remains to establish what can be learned from the amount of information collected in such a survey. One useful statistic that can be extracted from this data is the relative ranking of diseases: The option more often selected is the least prevalent one in the population, the second option in frequency is the second to last in prevalence and so on. Naturally, a statistical test such as χ^2 should be used to assess the significance of the ranking and expose random artifacts, also, the predisposition of someone negatively selecting an option over the others should be accounted for (it is not necessarily the case that every option that can be negatively selected is chosen with equal probability). The precise details of how to carry this out and an investigation of other statistics, such as estimating population frequencies, are the themes of current work.

5.4 Summary

In this chapter I have explored some properties of representing data negatively. The discussion is centered on a finite universe of items partitioned into two sets—the positive DB and the negative UDB—where the positive is deemed to contain the items of interest and is, in general, considered to be much smaller than its image. The characteristics examined pertain to the differences in the amount of information contained by negative and positive records, to how it changes as the membership of items becomes known, and to how answers are inferred using a positive set or its negative counterpart. Given the stated assumptions, the amount of information per string is lower in a negative set and increases as the identity of the items it collects is revealed (until a certain point, see Figure 5.2). Inferring answers from either set differs in that to establish that x belongs to DB, all UDB must be inspected, while the search may stop as soon as x is encountered using DB. The converse is true when x is a member UDB.

The analysis is first carried out while considering the sets DB and UDB which collect items without any efficiency considerations, it is then extended to negative databases, as outlined in Chapter 4, which more succinctly represent UDB. The current examinations make a series of assumptions that help glean the aforementioned properties but that need to be re-examined in future work, most important is the supposition that strings in DB are i.i.d. This is clearly not the case is a vast number of applications and the correlation between strings needs to be accounted for in order to have a more accurate idea of how information is allocated in the databases and to how knowledge of the membership of strings affects it.

The differences between representing data positively or negatively opens the door for novel applications and to a different implementation of some older ones. The last section of the chapter details a setup by which a negative database is distributed and, by exploiting some of its unique properties, is able to perform some operations privately: In one arrangement a personalized message may be posted in a public site that can only be retrieved by its intended recipient, and only he can know it was reclaimed. The setup allows to query a database while concealing the answers from its administrator. Also, an information gathering scheme is proposed that allows one to compute certain statistical features of a population while safeguarding the privacy of the subjects. These schemes provide a measure of privacy that arises naturally from representing information negatively, rather that from the use of traditional cryptography, and in contrast with this, do not rely on a specific secret being kept or on a particular code being hard to break. It is important to have alternatives to traditional cryptographic methods for the cases in which they might be unavailable or impractical to use.

Chapter 6

Conclusion

In the 1940's a philosopher by the name of Carl G. Hempel [76] showed that by manipulating the logical statement “All ravens are black” one could derive the equivalent “ All non-black objects are non-ravens”, indicating that the presence of a non-black object, like a yellow pencil, is evidence for the original assertion. This somewhat counterintuitive and roundabout way of proving a statement is actually correct, albeit not always possible or feasible to execute. Think of smaller sets and the statement “All ravens in this zoo are black” and its equivalent “All non-black animals in this zoo are non-ravens”. Here, parading all non-black animals at the zoo is actually feasible and will serve to prove or disprove the assertion. In the same spirit, a set can be defined by its negative image; the inventory of every animal except black crows can be used as a surrogate for the set of black crows by simply specifying “whatever is not on this inventory,” provided that we have knowledge of what all the possible animals are.

This dissertation has focused on two questions regarding such a convoluted way of describing sets (and proving things about them), namely: Can it be done efficiently, and what can be gained from it? Theoretically, as long as the universe is finite, it is always possible to list all the objects not in some arbitrary subset but, as the crow example suggests, the list might be too big and it might take too long a time to generate. Therefore, can it be done efficiently? The second concern is more open-ended and at first glance, hard to imagine. However, it was the observance of a system that uses negative data representation that prompted me to investigate its nuances further.

The immune system (IS) is primarily responsible for keeping an organism healthy; for preventing the intrusion of disease causing agents. In order to eliminate potential pathogens the IS must first identify them, but how can agents be recognized if they have never been encountered before (as is the case of many pathogens for a given organism)? The IS solves the riddle by creating immune cells that do *not* recognize *self*—self being the normal constituents of the organism [1]. This is much like the crow example, only that the assertion is "All self agents are good" and its contrapositive—the one that supports the IS's strategy— is "All non-good agents (pathogens) are non-self". Identifying non-self ensures that pathogens will be recognized. We can think of all the immune cells of an organism as the negative image of self, as collectively defining self by individually specifying what it is not.

The IS example gives an answer to the "what for" part of the question but offers only a partial answer to the feasibility aspect of storing the negative image of a set. The reason being that the IS does not exactly recognize non-self, that is, immune cells are created based on a sample of what self is and generalize from it to what self is likely to be; hence significantly reducing the size of the purported non-self and making it practical to store. Chapter 3 examines how to generalize from a sample dataset to a probable one. It discusses how the resulting set can be depicted either positively or negatively and analyses some of the tradeoffs between representations. The scheme proposed therein is presented in the context of anomaly detection (owing

[1] Danger theory [96] offers a different avenue for antigen recognition.

to its inspiration from the IS) and suggests a connection with database theory, with the expectation that insights from one field will illuminate the other.

But, what if what we want is to create an exact negative image without any generalization. What if "self" is actually a database like an address book? Can a negative image of it be created efficiently? Chapter 4 shows, surprisingly, that this is feasible by introducing the concept of negative databases—a negative database is a special representation of the negative image of a set—and provides algorithms for creating them, as well as for performing basic operations like the insertion and deletion of items. Furthermore, the chapter draws out an important characteristic of certain kinds of negative databases, namely that they are easy to create, easy to query in specific ways but hard, indeed $\mathcal{NP}$-hard, to revert into their original positive counterpart. It is easy to verify if there is a specific person with a specific address in the positive address book (using its negative database) but hard, for example, to determine what are the names of the people with an address in Mexico City. It is even hard to glean how many addresses are recorded in the positive address book. This property alone can be useful for privacy and security applications, where specific searches should be efficiently supported while impeding arbitrary fishing expeditions. For example, set intersection (say two banks want to know which transactions they have in common) can be performed securely in a very natural fashion using hard to reverse negative databases.

Hempel, the philosopher mentioned in the beginning of the chapter, came up with a paradox [2] by looking at the statement "All ravens are white" and its contrapositive "All non-white objects are non-ravens". It seems now that a yellow pencil can be taken as evidence for two contradictory (and mutually exclusive) assertions. The pencil simultaneously supports that all ravens are black and that all ravens are white. Chapter 5 studies how knowledge regarding the composition of a set, i.e. which items

[2] The paradox is more of a comment on the way scientific endeavors proceed. In particular, in regards to the importance and definitiveness of evidence.

it is likely to contain, changes as negative and positive evidence is presented. It shows that a negative witness does indeed contain information and that observing a yellow pencil increases the likelihood of either claim being true, without indicating which one; a feature that can be used for the selective disclosure of information. When the universe of discourse is finite, as in the case of this dissertation, only one (if any) will be known to be true once all evidence is examined, i.e. when all of the negative image is disclosed, doing away with the apparent paradox. This point illustrates a fundamental distinction in how answers are inferred using a positive set versus its negative counterpart. The latter part of Chapter 5 suggests how the differences can be used for the private dissemination of data and how certain population statistics can be gathered with the aid of these insights.

The chief contribution of the dissertation has been to draw attention to an alternate way to represent data, one that is often alluded to in our everyday discourse (did you notice the president didn't mention X in his speech?) but hasn't been studied in a rigorous fashion. The discussion was limited to specific kinds of sets (consisting of finite length strings) and showed that, even here, there are some distinctive characteristics in representing information negatively and that it is actually feasible to create and store them (think of how many possible names and addresses are not in your address book!).

There are many venues for future work beyond refining the current analysis (suggestions are provided at the end of each chapter). The present research draws out only a few distinguishing characteristics of negative representations, more should be sought out. I discussed how my scheme can be used to keep, share and consult data privately and briefly suggested how it may be employed to collect information in a private fashion. It would be fruitful to pursue these applications further as well as to search for new ones. For instance, what are the advantages of extracting knowledge from a negative set, is negative data mining a useful possibility? Moreover, it would be of interest to study alternative encodings of negative information and expand their scope beyond finite sets of strings. Finally, I would like to go back to the themes that inspired this monograph, biology and the immune system, and see if the lessons learned and the tools developed here shed light to the workings of these intricate systems. Can the immune system be thought of as a negative database that is hard to reverse, as a security system that co-evolves with the pathogens that try to subvert it?

Appendix A

Detectors and Generalization: A Detailed Derivation

A.1 r-chunks Matching Subsumes rcb Matching

We say $L \subset U$ is a language recognized by $Scheme_{\alpha,\beta}$ using r-chunks [rcb] detectors if there exists a set Υ of r-chunks [rcb] detectors such that $Scheme_{\alpha,\beta}(\Upsilon) = L$. From the perspective of language recognition, but not necessarily efficiency, r-chunk detectors subsume rcb detectors as it is now demonstrated.

Lemma A.1.0.1. If d is an rcb detector, there exists a collection $T(d)$ of r-chunk detectors such that $\forall x \in U$, $d[w] = x[w] \leftrightarrow \exists\, t \in T(d)$, $t = x[w]$.

Proof. Take $T(d)$ to be $\bigcup_{w_i}(d[w_i])$ and the result follows immediately. □

It follows from the lemma that, for each of the four detection schemes defined is Section 3.2, any set $L \subset U$ of strings that can be recognized with a set of rcb

detectors can be recognized with some set of r-chunk detectors. In particular, we have:

Theorem A.1.0.1. If Υ is a set of *rcb* detectors, and α and β are any choices for detector dimensions as above, then $Scheme_{\alpha,\beta}(\Upsilon) = Scheme_{\alpha,\beta}(T(\Upsilon))$.

Proof. It follows from lemma A.1.0.1 that for any of the membership predicates $F_{\alpha,\beta}$ associated with any of the four detection schemes $Scheme_{\alpha,\beta}$, and for every string $x \in U$, [Υ satisfies $F_{\alpha,\beta}$ *wrt* x] $\leftrightarrow$ [$T(\Upsilon)$ satisfies $F_{\alpha,\beta}$ *wrt* x]. □

The converse of this theorem is not true in general, since r-chunk detectors have a finer granularity than do *rcb* detectors—a proper subset of $T(d)$ may match fewer strings in U than does the *rcb* d.

Example: Consider the detection scheme $Scheme_{ND}$, and let $l = 3$ and $r = 2$. Consider the pair of strings 011 and 010. I claim that whenever both strings 011 and 010 are in the language of $Scheme_{ND}(\Upsilon)$ for a set Υ of *rcb* detectors, then so too must be the string 110. To see this, note that an *rcb* detector matching 110 must either end in the pattern $*10$ or begin with the pattern $11*$. But then any way either of these patterns is completed (i.e., by specifying a bit for the $*$) results in an *rcb* detector matching one of 011 or 010. Consequently, if $Scheme_{ND}(\Upsilon)$ excludes 110 from the language it recognizes, it must exclude also at least one of 011 or 010. In contrast, if the r-chunks detector set, Υ_{ch}, consists of the single detector $11*$, $Scheme_{ND}(\Upsilon_{ch})$ includes both 011 and 010 while excluding 110.

The results of the next section imply that the same result holds for $Scheme_{PC}$, that is, that the class of languages recognized by $Scheme_{PC}$ using r-chunks detectors properly contains the class recognized by $Scheme_{PC}$ using *rcb* detectors.

A.2 Detector Set Size

In Section 3.4.1 the size of the detector set for the sliding window r-chunks decomposition is summarized. In this appendix I present the complete derivation.

Assume a sample S of strings is randomly generated. To provide an estimate of the average number of detectors that can be produced, for both the $Scheme_{PC}$ and $SchemeND$ detection schemes, note the following:

- The number of strings in U with a specific pattern in any given window of size r is $\mathbb{A}^{l-r}$, where $\mathbb{A}$ is the cardinality of the alphabet over which strings are defined.

- The probability of selecting at random one such string is $\frac{\mathbb{A}^{l-r}}{\mathbb{A}^{l}} = \mathbb{A}^{-r}$.

- Assuming r is small compared to l, I approximate the probability that a randomly generated self set of $|S|$ unique strings will contain a specific pattern by

  $$1 - \binom{|S|}{0}(\mathbb{A}^{-r})^{0}(1-\mathbb{A}^{-r})^{|S|} = 1 - (1-\mathbb{A}^{-r})^{|S|}$$

  The equation is approximate because it considers trials to be independent and sampling to take place with replacement.

- Following the previous item, the expected number of distinct patterns E_r, for a given window of size r, is approximated by $E_r \approxeq \mathbb{A}^{r} - \mathbb{A}^{r}(1-\mathbb{A}^{-r})^{|S|}$.

The following equation denotes the expected size of a set of detectors having the property that each member matches one window and all windows are matched:

$$E_{pos} = tE_r \tag{A.1}$$

Conversely, the expected number of detectors possible for the negative detection scheme (i.e. detectors that don't match any window pattern in S) is given by:

$$E_{neg} = t(\mathbb{A}^r - E_r) \tag{A.2}$$

To establish when one scheme requires a smaller set of detectors than the other for maximal coverage, I determine the number of strings in S for which both schemes yield the same number of detectors, i.e. when $E_{pos} = E_{neg}$ (see Figure 3.4):

$$E_{pos} = E_{neg} = t(\mathbb{A}^r(1 - \mathbb{A}^{-r})^{|S|}) = t(\mathbb{A}^r - \mathbb{A}^r(1 - \mathbb{A}^{-r})^{|S|})$$

Solving for $|S|$

$$|S| = \frac{-\ln(2)}{\ln(1 - \mathbb{A}^{-r})} \approxeq (0.693)\mathbb{A}^r \tag{A.3}$$

A.2.1 Reduced Detector Set for Negative Detection

The number of detectors needed in $Scheme_{ND}$, to protect $CC(S)$, can actually be smaller than the full set described above, since significant redundancy amongst detectors may be present. I examine this property in detail for the case of a binary alphabet. Number the t windows from left to right:

- Regardless of the composition of S, a detector must be generated for each pattern in the first window that is not present is S.
- For every window of size r, starting from the second window, and for every pair of patterns $w1 = v_i \ldots v_{r+i-2}a$, $w2 = v_i \ldots v_{r+i-2}\bar{a}$ in such a window:
  - If both $w1$ and $w2$ are present in self then we cannot generate any detector for them.

- If neither is present then we need not generate detectors for them since the preceding window will be missing its prefix, i.e. $bv_i \dots v_{r+i-2}$ or $\bar{b}v_i \dots v_{r+i-2}$ and a detector in the previous window will also match strings with such a pattern.
- If only one is present, say $v_i \dots v_{r+i-2}a$, then we must generate detector $v_i \dots v_{r+i-2}\bar{a}$ since no string in S contains it.

With this in mind, the average number of detectors needed in the minimal set is given by:

$$E_{minN} = 2^r - E_r + (l - r)(E_r - 2(E_r - E_{r-1})) \tag{A.4}$$

Following the previous rationale an upper bound can be established for the number of detectors required in the minimal set. The maximum number of self strings we can have without creating crossovers, thereby reducing the number of required detectors, will exhibit only one of every pair of patterns $w1 = v_i \dots v_{r+i-2}a$, $w2 = v_i \dots v_{r+i-2}\bar{a}$ for each window. This results from a maximum of 2^{r-1} distinct self strings for a detector set size of $t2^{r-1}$. Figure 3.4 shows the plots of the expected number of detectors for both the full and reduced detector sets.

A.2.2 Reduced Detector Set Size for Positive Detection

Using similar reasoning to that of Section A.2.1, we can find a significant amount of redundancy amongst detectors in the form of *implicit matches.* Consider the case where window $i+1$ contains patterns $w1 = v_i \dots v_{r+i-2}a$, $w2 = v_i \dots v_{r+i-2}\bar{a}$, then it must be that window i has either or both of the patterns ending with bits $v_i \dots v_{r+i-2}$ and therefore, a string matched in window i by one of these patterns, will also be matched in window $i+1$ by either $w1$ or $w2$. I call such a match an *implicit match* and eliminate $w1$ and $w2$ from the detector repertoire. The number of detectors in the resulting set can be expressed as:

$$E_{minP} = E_r + (l - r)(E_r - 2(E_r - E_{r-1})) \tag{A.5}$$

The size of the sample S for which E_{minN} and E_{minP} yield the same number of detectors is the same as with the full repertoire, i.e. E_{neg} and E_{pos} eq.(A.3).

One subtlety about this analysis is that if not all positive detectors are represented explicitly, then some additional information is required to identify implicit matches. This could be stored explicitly, requiring at the very least one bit per implicit match or, it could be derived at runtime by determining, once a match (or implicit match) at window i has been established, if both $w1$ and $w2$ are absent in window $i + 1$ [1]. A plot of E_{minP} for different sample sizes is presented in figure 3.4 without regards to the extra information required to determine implicit matches.

A.3 The Crossover Closure Expected Size

In order to determine the size of CC for a randomly generated sample of binary strings, note that the number of substrings of length l that contain pattern $v_i \dots v_{r+i-1}$ in window w_i is double the number of substrings of length $l-1$ that contain the pattern in the same window if there are strings in S that exhibit both $v_{i+1} \dots v_{r+i-1}a$ and $v_{i+1} \dots v_{r+i-1}\bar{a}$ in window w_{i+1}, and stays the same if exactly one of these is present. In terms of the DAG representation (Figure 3.5), the number of paths that include a node with a given label doubles when such a node has two outgoing edges.

[1] If the information is stored explicitly the extra bits can be "recovered" as a decrease in execution time.

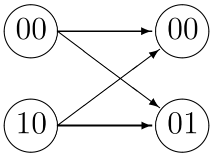
We can write this as a recurrence on the number of windows t:

$$CC(t) = \begin{cases} E_r & if \quad t = 1 \\ 2CC(t-1)P(2) & \\ +CC(t-1)(1-P(2)) & otherwise \end{cases}$$

Solving the recurrence yields:

$$CC(t) = E_r(1 + P(2))^{(t-1)} \tag{A.6}$$

where $P(2)$ is the probability of a node having two outgoing edges. In order establish the value for $P(2)$ consider the following gadget:



The probability for a given edge to be present is approximated by $\frac{E_{r+1}}{2^{r+1}} = 1 - \left(1 - 2^{-(r+1)}\right)^{|S|}$ and its absence by $\frac{2^{r+1} - E_{r+1}}{2^{r+1}} = \left(1 - 2^{-(r+1)}\right)^{|S|}$.

Given that the likelihood of a node having only one outgoing edge is not independent of the probabilities related to the second node in the graph (a node at the same level differing only in the first bit position), I consider the probability $P^*(1)$ of either node having one outgoing edge:

$$P^*(1) = 4\frac{E_{r+1}}{2^{r+1}}\left(\frac{2^{r+1} - E_{r+1}}{2^{r+1}}\right)^2$$

Similarly, for a node in the gadget to have no outgoing edges (or for the node to be absent):

$$P^*(0) = \left(\frac{2^{r+1} - E_{r+1}}{2^{r+1}}\right)^4$$

Finally, the probability of an individual node having two outgoing edges is given by:

$$P(2) = 1 - \frac{1}{2}(P^*(0) + P^*(1)) \tag{A.7}$$

The number of holes $|H|$ (strings deemed to be a part of self (RS) not present in S) can be derived by simply subtracting $|S|$ from $CC(t)$:

$$|H| = CC(t) - |S| \tag{A.8}$$

Appendix B

Negative Databases: Definitions and Proofs

Definition B.0.0.1. *c-clause*: A c-clause is a pattern composed of a $\bar{c}$-key (see Definition 4.4.1.2) plus at most two additional specified bit positions.

Lemma B.0.0.1. The Randomize_NDB algorithm, under any sequence of random choices, produces an NDB that corresponds to an instance of SAT that is satisfied exactly by DB.

Proof. Let ns_j be any string in $U - DB$ and let i be the length of the smallest prefix V_p of ns_j that is absent from DB. The algorithm will find this prefix at iteration i (line 3) and create at least one distinct string with a subpattern p of V_p that is absent from DB (steps 4–11).

If p is not found in DB then p must be different in at least one bit form every pattern in DB and $\bar{p}$ must match every string in DB in at least one position. The mapping to SAT creates clauses that correspond to $\bar{p}$ (see Figure 4.5 and lemma B.0.0.1) and are thus satisfied by every string in DB and unsatisfied by ns_j (for all

$ns_j \in U - DB$).

□

Lemma B.0.0.2. The $RNDB$ algorithm can generate any formula of at most n c-clauses containing solely the n variables present in window w_n (when w_n is the first window considered) that is satisfied exactly by DB^α, where DB^α consists of all the n length prefixes of the strings in DB.

Proof. Let ϕ be a formula satisfied exactly by DB^α and $C_1 \ldots C_n$ the c-clauses composing ϕ. $U^\alpha - DB^\alpha$. For any c-clause C_q in ϕ, the complement pattern does not satisfy it. By definition, any string containing the complement of C_q is in $U^\alpha - DB^\alpha$ and every string containing the complement pattern of C_q is considered by the algorithm, and can generate C_q. Note that each call to Pattern_Generate (Fig. 4.5) returns only one clause. However, up to $n \leq l$ calls are made on the same V_p, so even if all n clauses in ϕ must come from the same V_p, there are sufficient calls to account for them.

Further, no clause not in ϕ need be generated when w_n is considered because every string s in $U^\alpha - DB^\alpha$ that is considered for window w_n must contain (since it does not satisfy ϕ) the complement pattern of at least one C_q of ϕ and thus is capable, with an appropriate sequence of random choices, of generating this C_q and no additional clause (clauses generated repeatedly appear only once in the set of clauses returned).

Note that if there exists one or more formulas of at most n c-clauses containing solely the first n variables which are satisfied by exactly DB^α, $RNDB$ will add no additional clauses after the initial window w_n is considered, because at future iterations there will be no strings which do not appear in w_{i+1} that have a prefix in w_i.

□

Theorem B.0.0.1. The $RNDB$ algorithm can generate every possible 3-SAT formula such that the number of clauses is bounded by the number of variables.

Proof. Let ϕ be any 3-SAT formula of l variables and let DB be the set of assignments that exactly satisfy ϕ.

For every database DB there exists another database, DB^{β}, such that DB contains all the l-length prefixes of strings in DB^{β} and DB^{β} contains every possible string of length 2^l with those prefixes. The $RNDB$ algorithm on input DB^{β} will set its initial window to encompass the first l bit positions, by lemma B.0.0.2 the algorithm can generate any formula of at most l c-clauses containing only the first l bit positions of DB^{β}. After considering this first window the algorithm will not generate any more clauses, since there are no additional strings in $U^{\beta} - DB^{\beta}$ whose immediate prefix is contained in DB^{β}. Hence the $RNDB$ algorithm will output ϕ by making the appropriate random choices. □

Corollary B.0.0.1. The image of $RNDB$ defines an $\mathcal{NP}$-complete restriction of NESR. Similarly, the image of $RNDB$ defines an $\mathcal{NP}$-hard restriction of ESR and SR.

Proof. By Theorem B.0.0.1 $RNDB$ can generate an NDB corresponding to (under the transformation of the proof of Theorem 4.2.0.2) any instance of 3-SAT in which the number of clauses is bounded by the number of variables. The set of all such instances of 3-SAT is known to define an $\mathcal{NP}$-complete problem. □

References

[1] D. Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating satisfiable problem instances. In *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, pages 256–261, Menlo Park, CA, July 30– 3 2000. AAAI Press.

[2] D. Achlioptas, H. Jia, and C. Moore. Hiding satisfying assignments: Two are better than one. In *AAAI*, pages 131–136, 2004.

[3] N. R. Adam and J. C. Wortman. Security-control methods for statistical databases. *ACM Computing Surveys*, 21(4):515–556, December 1989.

[4] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *CIDR*, pages 186–199, 2005.

[5] D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Symposium on Principles of Database Systems*, 2001.

[6] R. Agrawal, T. Imielinski, and A. Swami. Mining associations between sets of items in massive databases. In *Proc. of the ACM-SIGMOD 1993 Int'l Conference on Management of Data*, pages 207–216, Washington D.C., May 1993.

[7] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1995.

[8] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 439–450. ACM Press, May 2000.

[9] D.W. Aha, D. Kibler, and M.K. Albert. Instance-based learning algorithms. *Machine Learning*, (6):37–66, 1991.

[10] A. V. Aho, Y. Sagiv, and J.D. Ullman. Equivalence of relational expressions. *SIAM J. Comput.*, (8):218–246, 1979.

[11] M. Ayara, J. Timmis, R. de Lemos, L. N. de Castro, and R. Duncan. Negative selection: How to generate detectors. In J Timmis and P J Bentley, editors, *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)*, pages 89–98, University of Kent at Canterbury, September 2002. University of Kent at Canterbury Printing Unit.

[12] T. Back, D. B. Fogel, and Z. Michalewicz (Editors). *Handbook of Evolutionary Computation.* Oxford University Press.

[13] J. Balthrop, F. Esponda, S. Forrest, and M. Glickman. Coverage and generalization in an artificial immune system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-02)*, pages 3–10, 2002.

[14] E.R. Bareiss, B. Porter, and C.C. Weir. Protos: An exemplar-based learning apprentice. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 12–23, Irvine, CA, 1987. Morgan-Kaufmann.

[15] J. Cohen Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Advances in Cryptology—EUROCRYPT '93*, pages 274–285, 1994.

[16] C.M. Bishop. *Neural Nets for Pattern Recognition.* Oxford University Press, 1995.

[17] G. R. Blakley. Safeguarding cryptographic keys. In Richard E. Merwin, Jacqueline T. Zanca, and Merlin. Smith, editors, *1979 National Computer Conference: June 4–7, 1979, New York, New York*, volume 48 of *AFIPS Conference proceedings*, pages 313–317, Montvale, NJ, USA, 1979. AFIPS Press.

[18] G. R. Blakley and C. Meadows. A database encryption scheme which allows the computation of statistics using encrypted data. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 116–122. IEEE CS Press, 1985.

[19] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In George Robert Blakely and David Chaum, editors, *Advances in Cryptology: proceedings of CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*, pages 289–302, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1985. Springer-Verlag.

[20] D. W. Bradley and A. M. Tyrrell. The architecture for a hardware immune system. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *The Third NASA/DoD Workshop on Evolvable Hardware*, pages 193–200, Long Beach, California, 12-14 July 2001. IEEE Computer Society.

[21] D. W. Bradley and A. M. Tyrrell. A hardware immune system for benchmark state machine error detection. In *Proceedings of the Congress on Evolutionary Computation 2002 (CEC2002)*, Honolulu, Hawaii, May 2002.

[22] D. W. Bradley and A. M. Tyrrell. Immunotronics: Novel finite state machine architectures with built in self test using self-nonself differentiation. *IEEE Transactions on Evolutionary Computation*, 6(3):227–238, June 2002.

[23] G. Brassard. A note on the complexity of cryptography. *IEEE Transactions on Information Theory*, 25(2), 1979.

[24] G. Brassard, S. Fortune, and J. E. Hopcroft. A note on cryptography and NP$\cap$ coNP-P. Technical Report TR78-338, Cornell University, Computer Science Department, April 1978.

[25] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, 1984.

[26] R. Brinkman, J. Doumen, and W. Jonker. Using secret sharing for searching in encrypted data. In *Secure Data Management*, pages 18–27, 2004.

[27] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[28] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *Advances in Cryptology – CRYPTO ' 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 61–76. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2002.

[29] D. L. Chao and S. Forrest. Generating biomorphs with an aesthetic immune system. In Russell Standish, Mark A. Bedau, and Hussein A. Abbass, editors, *Artificial Life VIII: Proceedings of the Eighth International Conference on the Simulation and Synthesis of Living Systems*, pages 89–92, Cambridge, Massachusetts, 2003. MIT Press.

[30] F. Chin. Security problems on inference control for sum, max, and min queries. *J. ACM*, 33(3):451–464, 1986.

[31] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the Association for Computing Machinery*, 45(6):965–981, November 1998.

[32] S. A. Cook and D. G. Mitchell. Finding hard instances of the satisfiability problem: A survey. In Du, Gu, and Pardalos, editors, *Satisfiability Problem: Theory and Applications*, volume 35 of *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–17. American Mathematical Society, 1997.

[33] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[34] J. M. Crawford and L. D. Anton. Experimental results on the crossover point in satisfiability problems. In Richard Fikes and Wendy Lehnert, editors, *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 21–27, Menlo Park, California, 1993. American Association for Artificial Intelligence, AAAI Press.

[35] D. Dasgupta, editor. *An agent based architecture for a computer virus immune system*. GECCO 2000 Workshop on Artificial Immune Systems, 2000.

[36] D. Dasgupta and N. Attoh-Okine. Immunity-based systems: A survey. In *ICMAS workshop on "Immunity Based Systems*, 1996.

[37] D. Dasgupta and F. Gonzalez. An immunity-based technique to characterize intrusions in computer networks. *IEEE Transactions on Evolutionary Computation*, 6(3), June 2002.

[38] L. Nunes de Castro and F. J. Von Zuben. The clonal selection algorithm with engineering applications. In *Proc. of the Genetic and Evolutionary Computation Conference*, pages 36–39, Las Vegas, Nevada, USA, 8 2000.

[39] L.N. de Castro and J.I. Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer-Verlag, 2002.

[40] L. Dehaspe. *Frequent Pattern Discovery in First-Order Logic*. PhD thesis, Katholieke Universiteit Leuven, 1998.

[41] D. Denning. *Cryptography and Data Security*. AddisonWesley, Reading, MA, 1982.

[42] D.E. Denning and J. Schlorer. Inference controls for statistical databases. *Computer*, 16(7):69–82, July 1983.

[43] P. D'haeseleer. An immunological approach to change detection: theoretical results. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.

[44] P. D'haeseleer, S. Forrest, and P. Helman. An immunological approach to change detection: algorithms, analysis and implications. In *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*. IEEE Press, 1996.

[45] D. Dobkin, A. Jones, and R. Lipton. Secure databases: Protection against user influence. *ACM Transactions on Database Systems*, 4(1):97–106, March 1979.

[46] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[47] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley and Sons, 2001.

[48] D. Dasgupta (Editor). *Artificial Immune Systems and Their Applications*. Springer-Verlag, 1999.

[49] F. Esponda, E. S. Ackley, S. Forrest, and P. Helman. On-line negative databases. In Giuseppe Nicosia, Vincenzo Cutello, Peter J. Bentley, and Jon Timmis, editors, *Proceedings of the 3rd International Conference on Artificial Immune Systems (ICARIS)*, pages 175–188, Catania, Sicily, Italy, Sep 2004. Springer-Verlag.

[50] F. Esponda, E. S. Ackley, S. Forrest, and P. Helman. On-line negative databases (with experimental results). *International Journal of Unconventional Computing*, 1(3), 2005.

[51] F. Esponda and S. Forrest. Defining self: Positive and negative detection. Technical Report TR-CS-2002-02, University of New Mexico, 2002.

[52] F. Esponda and S. Forrest. Detector coverage under the r-contiguous bits matching rule. Technical Report TR-CS-2002-03, University of New Mexico, 2002.

[53] F. Esponda, S. Forrest, and P. Helman. The crossover closure and partial match detection. In Jonathan Timmis, Peter J. Bentley, and Emma Hart, editors, *Proceedings of the 2nd International Conference on Artificial Immune Systems (ICARIS)*, pages 249–260, Edinburgh, UK, Sep 2003. Springer-Verlag.

[54] F. Esponda, S. Forrest, and P. Helman. Enhancing privacy through negative representations of data. *Technical report, University of New Mexico*, 2004.

[55] F. Esponda, S. Forrest, and P. Helman. A formal framework for positive and negative detection schemes. *IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics*, 34(1):357–373, 2004.

[56] F. Esponda, S. Forrest, and P. Helman. Negative representations of information. *Submitted to International Journal of Information Security*, 2004.

[57] S. Even and Y. Yacobi. Cryptography and np-completeness. In *Proc. 7th Colloq. Automata, Languages, and Programming (Lecture Notes in Computer Science)*, volume 85, pages 195–207. Springer-Verlag, 1980.

[58] J. D. Farmer, N. H. Packard, and A. S. Perelson. The immune system, adaptation, and machine learning. *Physica D*, 22:187–204, 1986.

[59] J. Feigenbaum, E. Grosse, and J. A. Reeds. Cryptographic protection of membership lists. 9(1):16–20, 1992.

[60] J. Feigenbaum, M. Y. Liberman, and R. N. Wright. Cryptographic protection of databases and software. In *Distributed Computing and Cryptography*, pages 161–172. American Mathematical Society, 1991.

[61] C. Fiorini, E. Martinelli, and F. Massacci. How to fake an RSA signature by encoding modular root finding as a SAT problem. *Discrete Appl. Math.*, 130(2):101–127, 2003.

[62] S. Forrest, B. Javornik, R. Smith, and A. Perelson. Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation*, 1(3):191–211, 1993.

[63] S. Forrest, A. S. Perelson, L. Allen, and R. CheruKuri. Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[64] K. Fukunaga. *Introduction to Statistical Pattern Recognition (2nd Edition).* Academic Press, San Diego, CA, 1990.

[65] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice-Hall, 2001.

[66] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. pages 151–160, 1998.

[67] O. Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, 2000.

[68] S. Goldwasser. Multi party computations: past and present. In *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, pages 1–6. ACM Press, 1997.

[69] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. See also preliminary version in 14th STOC, 1982.

[70] F. Gonzalez, D. Dasgupta, and J. Gomez. The effect of binary matching rules in negative selection. In *Proceedings of the Genetic and Evolutionary Computation Conference(GECCO)*, Chicago, July 2003.

[71] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining.* MIT Press, 2001.

[72] S. J. Hanson and J. Kegl. Parsnip: A connectionist network that learns natural grammar from exposure to natural language sentences. In *Proceedings of the Ninth IAnnual Conference on Cognitive Science*, 1987.

[73] P. K. Harmer, P. D. Williams, G. H. Gunsch, and G. B. Lamont. An artificial immune system architecture for computer security applications. *IEEE Trans. Evolutionary Computation*, 6(3):252–280, 2002.

[74] P.E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:515–516, 1968.

[75] P. Helman. *The Science of Database Management.* Richard D. Irwin, Inc., 1994.

[76] C. G. Hempel. Studies in logic and confirmation. *Mind*, 54:1–26, 1945.

[77] S. Hofmeyr. *An immunological model of distributed detection and its application to computer security.* PhD thesis, University of New Mexico, Albuquerque, NM, 1999.

[78] S. Hofmeyr and S. Forrest. Immunity by design: An artificial immune system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1289–1296, San Francisco, CA, 1999. Morgan-Kaufmann.

[79] S. Hofmeyr and S. Forrest. Architecture for an artificial immune system. *Evolutionary Computation Journal*, 8(4):443–473, 2000.

[80] S. Hofmeyr, A. Somayaji, and S. Forrest. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6:151–180, 1998.

[81] J.H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.

[82] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 12–24. ACM Press, 1989.

[83] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. In IEEE, editor, *30th annual Symposium on Foundations of Computer Science, October 30–November 1, 1989, Research Triangle Park, NC*, pages 236–241, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1989. IEEE Computer Society Press.

[84] C. A. Janeway, P. Travers, and M. Walport. *Immunobiology.* Garland Publishing, 1999.

[85] N. Japkowicz. Are we better off without counter-examples? In *Proceedings of the First International ICSC Congress on Computational Intelligence Methods and Applications (CIMA-99)*, pages 242–248, 1999.

[86] N. Japkowicz. *Concept-Learning in the Absence of Counter-Examples: An Autoassociation-Based Approach to Classification.* PhD thesis, University of New Jersey, New Brunswick, NJ, 1999.

[87] M. Karnaugh. The map method for synthesis of combinational logic circuits. *Trans. AIEE*, 1953.

[88] J. Kim and P. J. Bentley. An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1330–1337, San Francisco, CA, 2001. Morgan-Kauffman.

[89] P. Koton. Reasoning about evidence in causal explanations. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 256–261, St. Paul, MN, 1988. Morgan-Kaufmann.

[90] Z. Li, A. Das, and J. Zhou. Theoretical basis for intrusion detection. In *Proceedings of 6th IEEE Information Assurance Workshop (IAW)*, West Point, NY, June 2005.

[91] M. L. Littman. Algorithms for sequential decision making. Technical Report CS-96-09, 1996.

[92] K. Nissim M. Freedman and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology – Eurocrypt '2004 Proceedings, LNCS 3027*, pages 1–19. Springer-Verlag, May 2004.

[93] M. V. Mahoney and P. K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 376–385, New York, NY, USA, 2002. ACM Press.

[94] D. Maier. *The Theory of Relational Databases.* Computer Science Press, 1983.

[95] N. S. Matloff. Inference control via query restriction vs. data modification: a perspective. In *on Database Security: Status and Prospects*, pages 159–166. North-Holland Publishing Co., 1988.

[96] P. Matzinger. The danger model: A renewed sense of self. *Science*, 296(5566):301–305, 2002.

[97] E.J. McCluskey. Minimization of boolean functions. *Bell System Technical Journal,*, 1956.

[98] R. C. Merkle and M. E. Hellman. Hiding information and signatures in trapdoor knapsacks. IT-24:525–530, 1978.

[99] S. Micali, M. Rabin, and J. Kilian. Zero-knowledge sets. In *Proc. FOCS 2003.*, 2003.

[100] D. Mitchell, B. Selman, and H. Levesque. Problem solving: Hardness and easiness - hard and easy distributions of SAT problems. In *Proceeding of the 10th National Conference on Artificial Intelligence (AAAI-92), San Jose, California*, pages 459–465. AAAI Press, Menlo Park, California, USA, 1992.

[101] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, USA, 1998.

[102] T. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[103] T.M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[104] R. Morselli, S. Bhattacharjee, J. Katz, and P. Keleher. Trust preserving set operations. Technical report.

[105] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing: Seattle, Washington, May 15–17, 1989*, pages 33–43, New York, NY 10036, USA, 1989. ACM Press.

[106] B. Nauman. A cast of the space under my chair. concrete, 1965-8.

[107] M. Negri, G. Pelagatti, and L. Sbattella. Formal semantics of sql queries. *ACM Transactions on Database Systems (TODS)*, 16(3):513–534, 1991.

[108] A. M. Odlyzko. The rise and fall of knapsack cryptosystems. In *PSAM: Proceedings of the 42th Symposium in Applied Mathematics, American Mathematical Society*, 1991.

[109] R. Ostrovsky, C. Rackoff, and A. Smith. Efficient consistency proofs for generalized queries on a committed database. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 2004.

[110] J. K. Percus, O. Percus, and A. S. Perelson. Probability of self-nonself discrimination. In A. S. Perelson and G. Weisbuch, editors, *Theoretical and Experimental Insights into Immunology*, NY, 1992. Springer-Verlag.

[111] J. K. Percus, O. Percus, and A. S. Perelson. Predicting the size of the antibody combining region from consideration of efficient self/non-self discrimination. *Proceedings of the National Academy of Science*, 90:1691–1695, 1993.

[112] W. V. Quine. A way to simplify truth functions. *American Mathematical Monthly*, 1955.

[113] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[114] D. E. Rumelhart, G. E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing*, pages 318–364. MIT Press, Cmbridge, MA, 1986.

[115] P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information (abstract). In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, page 188. ACM Press, 1998.

[116] S. Sathyanath and F. Sahin. Artificial immune systems approach to a real time color image classification problem. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 2001.

[117] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley and Sons, Inc., New York, NY, USA, 1994.

[118] A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[119] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts (Fourth Edition).* Mc Graw Hill, 2002.

[120] S. Singh. Anomaly detection using negative selection based on the r-contiguous matching rule. In Jonathan Timmis and Peter J. Bentley, editors, *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)*, pages 99–106, University of Kent at Canterbury, September 2002. University of Kent at Canterbury Printing Unit.

[121] L. Sompayrac. *How the Immune System Works.* Blackwell Science, 1999.

[122] T. Stibor, K. M. Bayarou, and C. Eckert. An investigation of r-chunk detector generation on higher alphabets. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund Burke, Paul Darwen, Dipankar Dasgupta, Dario Floreano, James Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andy Tyrrell, editors, *Genetic and Evolutionary Computation – GECCO-2004, Part I*, volume 3102 of *Lecture Notes in Computer Science*, pages 299–307, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.

[123] T. Stibor, P. Mohr, J. Timmis, and C. Eckert. Is negative selection appropriate for anomaly detection? In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 321–328, New York, NY, USA, 2005. ACM Press.

[124] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[125] D.W. Taylor and D.W. Corne. An investigation of the negative selection algorithm for fault detection in refrigeration systems. In Jonathan Timmis, Peter J. Bentley, and Emma Hart, editors, *Proceedings of the 2nd International Conference on Artificial Immune Systems (ICARIS)*, pages 34–45, Edinburgh, UK, sep 2003.

[126] P. Tendick and N. Matloff. A modified random perturbation method for database security. *ACM Trans. Database Syst.*, 19(1):47–63, 1994.

[127] J. Vaidya and C. Clifton. Secure set intersection cardinality with application to association rule mining. *To appear in Journal of Computer Security*, 2004.

[128] M.M Veloso and A. Aamodt (Eds.). Case-based reasoning research and development. In *Lecture notes in Artificial Intelligence*, Beriln, 1995. Springer-Verlag.

[129] P. Wayner. *Translucent Databases.* Flyzone Press, 2002.

[130] G. I. Webb. Efficient search for association rules. In *Knowledge Discovery and Data Mining*, pages 99–107, 2000.

[131] R. Whiteread. Untitled (library). Dental plaster, polystyrene, fiberboard and steel, 1999.

[132] S. T. Wierzchon. Discriminative power of the receptors activated by k-contiguous bits rule. *Journal of Computer Science and Technology*, 1(3):1–13, 2000.

[133] S. T. Wierzchon. Generating optimal repertoire of antibody strings in an artificial immune system. In M. A. Klopotek, M. Michalewicz, and S. T.Wierzchon, editors, *Intelligent Information Systems*, pages 119–133, Heidelberg New York, 2000. Physica-Verlag.

[134] S. T. Wierzchon. Deriving concise description of non-self patterns in an artificial immune system. In S. T. Wierzchon, L. C. Jain, and J. Kacprzyk, editors, *New Learning Paradigms in Soft Computing*, pages 438–458, Heidelberg New York, 2001. Physica-Verlag.

[135] P. D. Williams, K. P. Anchor, J. L. Bebo, G. H. Gunsch, and G. D. Lamont. CDIS: Towards a computer immune system for detecting network intrusions. In W. Lee, L. Me, and A. Wespi, editors, *Fourth International Symposium, Recent Advances in Intrusion Detection*, pages 117–133, Berlin, 2001. Springer.

[136] A. Yao. Protocols for secure computation. In IEEE, editor, *23rd annual Symposium on Foundations of Computer Science, November 3–5, 1982, Chicago, IL*, pages 160–164, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1982. IEEE Computer Society Press.