Enhancing Privacy through Negative Representations of Data (Paper ID 193)

Fernando Esponda
Stephanie Forrest
Paul Helman
Department of Computer Science
University of New Mexico
Albuquerque, NM 87131-1386
{fesponda,forrest,helman}@cs.unm.edu
Phone (505)2773112, Fax (505)2776927

December 8, 2003

Abstract

The paper introduces the concept of a negative database, in which a set of records DB is represented by its complement set. That is, all the records not in DB are represented, and DB itself is not explicitly stored. After introducing the concept, several results are given regarding the feasibility of such a scheme and its potential for enhancing privacy. It is shown that a database consisting of n, l-bit records can be represented negatively using only $O(ln)$ records. It is also shown that membership queries for DB can be processed against the negative representation in time no worse than linear in its size, and that reconstructing the original database from a negative representation is an NP-hard problem.

1 Introduction

Large collections of data are ubiquitous, and the demands that will be placed on these collections in the near future are increasing. We expect them to be available when we need them; we expect them not to be available to malicious parties; the contents of the collections and the rules for accessing them must be continually updated; we would like to be able to search them in new ways, drawing inferences about large-scale patterns and trends; we want to be protected from the wrong kinds of inferences being made (as in racial profiling); and, eventually, we will want the ability to audit the uses to which our personal data are put. Although many of these problems are old, they must now be solved more quickly for larger and more dynamic collections of data.

In this paper we introduce an approach to representing data that addresses some of these issues. In our approach, the negative image of a set of data records is stored rather than the records themselves. Initially, we assume a universe U of finite-length records (or strings), all of the same length l and defined over a binary alphabet. We logically divide the space of possible strings into two disjoint sets: DB representing the set positive records (holding the information of interest), and $U - DB$ denoting the set of all strings not in DB. We assume that DB is uncompressed (each

record is represented explicitly), but we allow $U - DB$ to be stored in a compressed form called NDB. We refer to DB as the *positive* database and NDB as the *negative* database.

From a logical point of view either representation will suffice to answer questions regarding DB. However, the different representations may present different advantages. For instance, in a positive database, inspection of a single record provides meaningful information. However, inspection of a single (negative) record reveals little meaningful information about the contents of the original database. Because the positive tuples are never stored explicitly, a negative database would be much more difficult to misuse.

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.

In the following sections, we first show that implementing NDB is computationally feasible. We do this by introducing a representational scheme that requires $O(ln)$ negative records to represent a positive database consisting of n, l-bit records, and then giving an algorithm for finding such a representation efficiently from any finite DB. This negative representation supports simple membership queries[1]. We then investigate some of the implications of the negative scheme for privacy. In particular, we show that the general problem of recovering the positive database from our negative representation is NP-hard, and we present an algorithm for creating such irreversible negative representations. Finally, we review related work, discuss the potential consequences of our results, and outline areas of future investigation.

2 Representation

In order to create a database NDB that is reasonable in size, we must compress the information contained in $(U - DB)$ but retain the ability to make queries. We introduce one additional symbol to our binary alphabet, 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 and will match 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 positions." With this new symbol we can potentially represent large subsets of $U - DB$ with just a few entries.

For example, the set of strings $(U - DB)$ can be exactly represented by the set depicted by NDB, as follows:

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

The convention is that s is taken to be in DB iff s fails to match all the entries in NDB. This condition is fulfilled only if for every $ns_j \in NDB$, s disagrees with ns_j in at least one defined

[1] Although indexing schemes could be developed to support truly efficient membership queries, our current emphasis is on demonstrating the dichotomy between tractable and intractable queries.

position. The following section gives an algorithm that creates a negative database NDB from the positive image DB. We then discuss some of the characteristics of the resulting representation.

2.1 The Prefix Algorithm

In this section we present an algorithm as proof that (1) a negative database NDB can be constructed from any finite positive database, and (2) that NDB is of reasonable size. The prefix algorithm introduced here creates an NDB with certain properties, in particular, concerning its ease of reversibility. As we shall see, making DB easily recoverable from NDB is not always desirable. Consequently, in Section 3, we shall consider an algorithm with other properties.

Prefix algorithm
Let w_i denote a window of the first i bit positions in a string, i.e., a prefix window of size i.
1. Find the largest prefix window, w_{ini}, for which all possible bit patterns are present in DB. Let i be the length of w_{ini}.
2. Create a record for each bit pattern in DB not present in window w_{i+1} but which has as a prefix one of the patterns present in window w_i. Use such patterns as prefixes for records, filling the remaining positions with $'*'$s.
3. Add the new records to NDB.
4. Increment i by one and repeat to step two while $i < l$ (where l is the length of strings).

Figure 1: The Prefix algorithm.

Lemma 2.1.1. The prefix algorithm creates a database NDB that matches exactly those strings not in DB.

Proof. Step two finds every prefix not present in DB that has not already been inserted in NDB. It then appends every possible string with

that prefix to NDB (step 3). If a DB pattern is not present in window w_{i+1} and its prefix is not in w_i then it must have been inserted in NDB before, since w_{ini} contains the prefixes of every string in U (step 1). □

Theorem 2.1.1. The number of records of the negative data set $(U - DB)$ can be represented using $O(l|DB|)$ records.

Proof. For every window of size i there are at most $|DB|$ "negative" records created (step 2) and inserted in NDB (step 3). The number of windows is at most l (step 4) therefore, the number of negative records is $O(l|DB|)$. □

3 Reversibility

In section 2.1 we presented an algorithm for generating NDB that easily demonstrates the feasibility of a negative representation and is appropriate when hiding DB is not an issue. In this section we turn our attention to the goal of making DB hard to reconstruct. First we establish a result that

shows the representation is potentially difficult to reverse, and then we present an algorithm which indeed produces hard to reverse instances.

Reconstruction of DB from NDB is NP-hard in the following sense[2].

Definition 3.0.1. Self Recognition (SR):
INPUT: $U - DB$ represented by a collection NDB of length l bit strings, such that each string may contain any number of '*' (don't care) symbols, and a candidate self set DB.
QUESTION: Does NDB represent the self set DB?

We establish SR is NP-hard. Note that NDB is an arbitrary $U - DB$ set, we are not specifying how it was obtained. First we establish the NP-completeness of the following problem.

Definition 3.0.2. Non-empty Self Recognition (NESR)
INPUT: $U - DB$ set represented by a collection NDB of length l bit strings, such that each string may contain any number of $'*'$ (don't care) symbols.
QUESTION: Is DB nonempty? That is, is there some string in $U = \{0,1\}^l$ not matched by NDB.

Theorem 3.0.2. NESR is NP-complete.

Proof. NESR is clearly in NP. (Guess a string, verify it is not matched.)
The NP-completeness of $NESR$ is established by transformation from 3SAT. Start with instance **I** of 3SAT. Let X be the set of variables $\{x_i\}$, and suppose l is the number of variables. The constructed instance of ESR will be over length l strings. Each clause $\{L_i, L_j, L_k\}$ (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). Similarly for the other two literals L_j and L_k in this clause.

Claim there exists a truth assignment satisfying **I** if and only if there exists a string in $U = \{0,1\}^l$ not matched by NDB. 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 **I**

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

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

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

□

Corollary 3.0.1. NESR is NP-complete even if every member of NDB contains exactly 3 specified (non $'*'$) bits.

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

Corollary 3.0.2. Empty Self Recognition (ESR, the complement of NESR, answers YES if and only if NDB represents the empty set) is NP-hard.

[2]For historical reasons we sometimes refer to DB as self and $U - DB$ as nonself.

Proof. Trivial Turing transformation from NESR. □

Theorem 3.0.3. Self Recognition (SR, defined above) is NP-hard.

Proof. We've established this to be the case even when the candidate self set DB is empty, and even when every member of NDB contains exactly 3 specified (non $'*'$) bits. □

3.1 The Randomize_NDB Algorithm

The prefix algorithm presented in Section 2.1 is simple and demonstrates that a compact negative representation NDB can be obtained from DB. While we have demonstrated in Section 3 that the general problem of reversing a given set NDB to obtain DB is NP-hard, utilizing the simple prefix algorithm to obtain NDB from DB raises two concerns with the privacy provided by a negative representation: (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) which produces NDB from DB could be reproduced by an adversary, this could allow easy recovery of DB from NDB. (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 NP-hard set of instances of NESR.) In this section, we present a randomized algorithm which addresses both these concerns. In particular, we modify the prefix algorithm by introducing a sequence of random choices that enlarge the set of instances of NDB it can produce, so that the reversibility of the problem instances in the algorithm's image defines an NP-hard problem. Further, since the execution of the algorithm is randomized, re-application of the algorithm by an adversary requires reproducing the algorithm's random sequence of choices.

Π−**Prefix algorithm**
1. Find a random permutation π and apply it to DB
2. NDB'=Run the prefix algorithm on $\pi(S)$.
3. NDB=Apply Π' to NDB', Π' is the inverse permutation of Π.

Figure 2: The Π-Prefix algorithm.

Lemma 3.1.1. Let B denote an arbitrary set of bit positions and let $p[B]$ denote a bit pattern over B.
For any subset B of the l bit positions containing the i^{th} position, and any pattern $p[B]$ such that $p[B]$ is not contained in DB and $p[B-i]$ is in DB, there exists a permutation such that the Π−prefix algorithm (fig. 2) places in NDB a string ns with $ns[B] = p[B]$ and '*' in each bit position of ns not in B.

Proof. There exist a permutation π such that when applied to DB all the positions in B are contiguous and the original i^{th} position is mapped to position $|B|$, such that $\pi(B)$ occupies the first $|B|$ positions of strings in $\pi(DB)$. By construction, the prefix algorithm, when run on $\pi(DB)$, will create an entry in NDB for $p[B]$ given that $p[\pi(B)]$ is not in $\pi(DB)$ while $p[\pi(B-i)]$ is.

□

Randomize_NDB algorithm
Let w_i denote a window of i bits positions starting at position one, i.e. a prefix window of size i.
1. Find random permutation π and apply it to DB.
2. Find the largest prefix window, w_{ini}, for which all possible patterns are present in $\pi(DB)$. Let i be the number of bits in such a window.
3. if $|w_{ini}| < log_2(l)$ then
   a) Set w_{ini} to be the first $\lceil log_2(l) \rceil$ positions.
4. Let $i = |w_{ini}|$.
5. For each prefix V_p not present in window w_{i+1} but that has as prefix a pattern present in window w_i:
   a)Call procedure Pattern_Generate($\pi(DB), V_p, l$) (see fig. 4).
   b)Create a record using the patterns output from Pattern_Generate. and fill the remaining positions with "don't care" symbols $'*'$.
6. Insert the new records in NDB.
7. Increment i by one and return to step two as long as $i < l$.
8. Apply π' to NDB. Where π' is the inverse permutation of π.

Figure 3: The Randomize_NDB algorithm.

Lemma 3.1.2. For every possible size $k \leq n$ set of V_p subpatterns not in DB, there is some execution of Pattern_Generate (fig. 4) (with an appropriate sequence of random choices) that will generate exactly that set.

Proof. Let a c-key[3] denote a pattern not present in DB with no extraneous bits. Pattern V_p may contain several c-keys.
For every pattern V_p and a c-key k contained in V_p there a exists a permutation π such that k occupies the $|k|$ rightmost bit positions of $\pi(V_p)$. 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. It follows that since k is a c-key and occupies the $|k|$ rightmost positions of $\pi(V_p)$ that k is the pattern that will be found[4]. Steps 7–11 of the algorithm guarantee that every possible combination of n patterns containing k has a chance to be generated.

□

The reduction of SAT to NESR used in the proof of Theorem 3.0.2 establishes a natural bidirectional correspondence between SAT instances and NDB instances. In the following, when we speak of a correspondence between instances of the problems (or between SAT clauses and sets NDB), we are referring to this mapping. This mapping also equates the (positive) input database DB with satisfying truth assignments of the corresponding SAT instance.

Lemma 3.1.3. The Randomize_NDB algorithm produces an NDB that corresponds to an instance of SAT that is satisfied exactly by DB.

[3]This is the concept of candidate key as defined in Silberschatz[15].

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

Pattern_Generate
On input DB, V_p, n
1. Find a random permutation π.
2. for i=1 to $|V_p|$ do {
3. Determine whether $\pi(V_p)$ without its i^{th} bit is in $\pi(DB)$
4. if not in $\pi(DB)$ then {
5. $\pi(V_p) \leftarrow \pi(V_p) - i^{th}$ bit (eliminate the bit)
6. Keep track of the i^{th} bit in a set indicator vector (SIV) }}
Suppose there are m bits in SIV.
7. Randomly choose $1 \leq j \leq n$
8. if $j \geq 2^m$ then
9. Create all 2^m subpatterns
else {
10. Randomly select j distinct numbers form 1 to 2^m.
11. Create[a] the corresponding patterns V_k and return $\pi'(V_k)$.}

[a]Note: A number corresponds to an inclusion rule of the bits in SIV for a specific pattern where 0 indicates exclude and 1 indicates include.

Figure 4: Procedure Pattern_Generate.

Proof. If a pattern 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. Our mapping to SAT creates clauses that correspond to $\bar{p}$ and are thus satisfied by every string in DB.

Now suppose there is a string $ns_j \notin DB$. Let i be the length of the smallest prefix of ns_j that is absent from DB. The algorithm will find this prefix at iteration i (line 5) and create at least one string with a subpattern p that is absent from DB (steps 5a and 5b). By our mapping this will correspond to a clause that will be unsatisfied by ns_j and consequently render the whole formula unsatisfied. Only assignments drawn from DB will satisfy the formulas.

□

We note that the Randomize_NDB algorithm runs in time $O(l^2|DB|^2)$ by noting that procedure Pattern_Generate (fig. 4) runs it time $O(l|DB|)$ and is called a total of $O(l|DB|)$ times.

The Randomize_NDB algorithm potentially produces sets NDB corresponding to formulas with up to $O(l|DB|)$ clauses. For the purpose of establishing irreversibly, it suffices to consider only those NDB produced corresponding to formulas with up to only l clauses. In particular, we show that essentially all formula with at most l clauses that are satisfied exactly by the members of the input set DB are potentially produced. We say essentially all because there are some trivially redundant clauses that the algorithm won't generate, such as clauses that are a superset of other previously created clauses (see lemma 3.1.1) and clauses with repeated literals. We can safely exclude this types of clauses from any formula without altering the set of truth assignments which satisfy it. In what follows, when we refer to all possible SAT formulas it is understood that we exclude ones containing clauses such as these.

Theorem 3.1.1. The Randomize_NDB algorithm can produce every *cnf* formula with at most l clauses that is satisfied exactly by the members of DB.

Proof. It suffices to show that the algorithm can produce, for every i, every combination of at most n clauses in which L_i is the literal with the largest index,[5] and by noting that by lemma 3.1.3 every and only strings from DB will satisfy the resulting formulas.

At iteration i the algorithm considers only patterns that specify position i and lower. By lemma 3.1.1 the algorithm potentially considers every possible clause of i variables where L_i is the literal with the highest index. By lemma 3.1.2 the algorithm (step 5a) potentially finds every possible subclause that is also satisfied by exactly DB.

Consequently the algorithm considers every possible combination of at most $l|DB|$ clauses that include L_i as the largest indexed literal for every i that are satisfied exactly by DB.

□

It is easy to show that SAT restricted to instances in which the number of clauses is linear in the number of variables remains NP-complete. It follows from theorem 3.1.1 that if we apply the transformation of theorem 3.0.2 to the version of SAT restricted to instances in which the number of clauses is linear in the number of variables, we will obtain an NDB (which defines an instance of NESR) that can be produced by our algorithm. Consequently, we have the following.

Theorem 3.1.2. The self recognition problem NESR restricted to instances that can be produced by the Randomize_NDB algorithm is NP-complete.

Note that this result implies that, even if it is known that NDB is produced by application of the algorithm, so long as the sequence of random choices made by the algorithm is not known, self recognition (and the associated self reconstruction problems discussed previously) cannot be solved in time polynomial in the size of NDB, unless P = NP.

4 Related work

There are several areas of research that are potentially relevant to the ideas discussed in this paper. These include: encryption, privacy-preserving databases, privacy-preserving data-mining, query restriction and negative data.

An obvious starting point for protecting sensitive data is the large body of work on cryptographic methods, e.g., as described in [14]. Cryptography, however, is intended to conceal all information about the encrypted data, and it is therefore not conducive to situations in which we want to support some queries efficiently but not reveal the entire database. Some researchers have investigated how to combine cryptographic methods with databases [11, 10, 4], for example, by encrypting each record with its own key.

In *privacy-preserving data mining*, the goal is to protect individual privacy while still supporting data-mining, for example, the computation of aggregate statistical properties [3, 2, 1, 6, 8, 17, 16]. An example of this approach is to store information that contains important properties of the real data but none of the details, for example, preserving the relevant statistical distributions but obscuring the details about individual records. Our method is almost the reverse of this approach, in that we support efficient membership queries but higher-level queries are likely to be expensive.

Our method is also related to query restriction[13, 5, 6, 7, 16], where the query language is designed to support only the desired classes of queries. Although query restriction controls access

[5] i.e. where for every literal L_j in the clause $j \leq i$.

to the data by outside users, it cannot protect an insider with full privileges from inspecting individual records to retrieve information.

The term "negative data" sounds similar to our method, but is actually quite different. The deductive database model (e.g., [12] presents an excellent survey of the foundations of the model) supports in the intensional database (IDB) the negative representation of data. The objectives, mechanisms, and consequences here are quite different from our scheme. In a deductive database, traditional motivations for "negative data" include reducing space utilization, speeding query processing, and the specification and enforcement of integrity constraints.

To summarize, 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.

5 Discussion and Conclusions

In this paper we have established the feasibility of a new approach to representing information. Notably, we showed that negative representations are computationally feasible and that they are difficult to reverse. However, there are many important questions and issues remaining.

Which classes of queries can be computed efficiently and which cannot? Our initial results address two extremes—the case of testing membership for a specific, single record and the case of reconstructing the entire database. We would like to understand the computational complexity at points across the spectrum between these two extremes, as well as understanding what computational properties are desirable in a privacy-protecting context. A related question involves the costs of database updates under our representation? How expensive is it to inserting or deleting entries from the negative database?

Are there other better representations of NDB? Once we understand more completely the computational properties of our current representations, we may be able to devise other representations whose properties are more appropriate for some applications.

An important feature of NDB is its distributability in which the NDB is partitioned into disjoint sets, or fragments. In a distributed NDB, positive membership queries can be processed with no communication among the database fragments. We would like to study this property in more detail.

Finally, we are interested in inexact representations. The NDB representation is closely related to partial match detection [9] which has many applications in anomaly detection. We are interested in studying how those methods might be combined with NDB either for designing an adaptive query mechanism or for approximate databases.

6 Acknowledgments

The authors gratefully acknowledge the support of the National Science Foundation (CCR-0331580, CCR-0311686, and DBI-0309147), Defense Advanced Research Projects Agency (grant AGR F30602-00-2-0584), the Intel Corporation, and the Santa Fe Institute. F.E. will also like to thank Consejo Nacional de Ciencia y Tecnología (México) grant No. 116691/131686 for its financial support.

References

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

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

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

[4] 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.

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

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

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

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

[9] 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*, In press.

[10] Joan Feigenbaum, Eric Grosse, and James A. Reeds. Cryptographic protection of membership lists. 9(1):16–20, 1992.

[11] Joan Feigenbaum, Mark Y. Liberman, and Rebecca N. Wright. Cryptographic protection of databases and software. In *Distributed Computing and Cryptography*, pages 161–172. American Mathematical Society, 1991.

[12] H. Gallaire, J. Minker, and J. Nicolas. Logic and databases: a deductive approach. *Computing Surveys*, 16:1:154–185, 1984.

[13] 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.

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

[15] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts (Fourth Edition)*. Mc Graw Hill, 2002.

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

[17] Peter Wayner. *Translucent Databases*. Flyzone Press, 2002.