On-line Negative Databases

FERNANDO ESPONDA, ELENA S. ACKLEY, STEPHANIE FORREST, PAUL HELMAN

Department of Computer Science
University of New Mexico
Albuquerque, NM 87131-1386
{*fesponda,elenas,forrest,helman*}*@cs.unm.edu*

The benefits of negative detection for obscuring information are explored in the context of Artificial Immune Systems (AIS). AIS based on string matching have the potential for an extra security feature in which the "normal" profile of a system is hidden from its possible hijackers. Even if the model of normal behavior falls into the wrong hands, reconstructing the set of valid or "normal" strings is an $\mathcal{NP}$-hard problem. The data-hiding aspects of negative detection are explored in the context of an application to negative databases. Previous work is reviewed describing possible representations and reversibility properties for privacy-enhancing negative databases. New algorithms are presented which allow on-line creation, updates and clean-up of negative databases, some experimental results illustrate the impact of these operations on the size of the negative database. Finally some future challenges are discussed.

1 INTRODUCTION

A striking feature of the natural immune system is its use of negative detection in which *self* is represented (approximately) by the set of circulating lymphocytes that fail to match self. The negative-detection scheme has been used in several artificial immune system (AIS) applications, and the benefits of such a scheme have been explored in terms of the number of required detectors [24, 14, 13, 49, 50], success in distinguishing self from nonself [20, 16], and

the ease with which negative detection can be distributed across multiple locations. In this paper we explore a fourth interesting property of negative representations, namely their ability to hide information about self. This information hiding ability has interesting implications for intrusion detection as well as for applications in which privacy is a concern and where it may be useful to adopt a scheme in which only the negative representation is available for querying.

This paper extends results first presented in [15] which introduced the concept of a *negative database*. In a negative database, a collection of data is represented by its logical complement. The term *positive information* denotes the elements of the category or set of interest (e.g., self) while *negative information* denotes all the other elements of the universe. A negative database is then a representation of the negative information. In addition to introducing this concept, the previous paper showed that negative information can be represented efficiently (even though the negative image will typically be much larger than the positive image), that such a representation can be $\mathcal{NP}$-hard to reverse (thereby hiding the exact composition of self), and that simple membership queries can be computed efficiently. For instance, a query of the form "is string x in the database" can be answered promptly, while a request of the form "give me all the strings in the positive image that start with a 1" cannot. However, the paper did not show that common database operations (such as inserts and deletes) can be performed easily on the negative representation or that a negative database can be maintained dynamically. Section 4 presents new algorithms that address these matters.

Many AIS used for anomaly detection represent the entity to be protected as a set of strings and, in parallel with the immune system, identification of anomalies is performed by an alternate set of strings (known as detectors) and a match rule that are designed not to match elements of self. In this context, there may be an additional incentive for negative detection using negative databases. When the negative information is represented as discussed in [15], it is provably hard to infer the positive image even if all the negative information is available. In the context of anomaly detection and security this provides an extra level of protection since it prevents someone from hijacking the detector set, deriving from it the normal profile of the system, and using that information to devise new attacks.

Section 2 reviews earlier work that is generally relevant to the topic of negative information; Section 3 reviews previous work on negative databases, showing that a negative representation can be constructed which occupies only polynomially more space than its positive counterpart, while retaining

some interesting querying capabilities. Section 4 presents on-line algorithms, including how to initialize a negative database and how to perform updates. Section 5 presents a study on the impact of updates on the size of the negative database and introduces an operation aimed at controlling its growth. Section 6 considers the implications of our results.

2 RELATED WORK

Negative representations of data have had several proponents in the past, especially in art where artists like Whiteread and Escher have taken advantage of the so called figure-ground relationship. Examples can also be found in mathematics and statistics where sometimes it is easier to obtain an answer by looking at the complement of the problem we intend to solve and complementing the solution. For the purpose of this paper, however, we will review how negative representations of information have influenced the field of AIS.

As mentioned in the introduction, the natural immune system can be interpreted as representing data negatively. This observation has led to algorithm designs which take advantage of some of the inherent properties of negative representations. In particular, designers have taken advantage of the fact that negative detectors are more amenable to distributed implementations than positive detectors and that, for the purposes of anomaly detection, negative representations of data seem more natural. The negative selection algorithm whereby a set of negative detectors is created was introduced in [19]. Anomaly-detection systems based on these ideas can be found in [46, 3, 51, 32, 11, 26, 27, 28]. Other applications have also been proposed that range from image classification to recommender systems [33, 47, 25, 42, 46, 8, 5, 6]. We note that many AIS are not based on string matching and therefore are not directly affected by the results presented here; the interested reader is referred to [12, 45].

Protecting information stored in a database from examination by unauthorized parties has been a concern since the inception of the field [41, 40, 44]. Some approaches relevant to the current discussion involve cryptographic protection of databases [17, 43, 48], multi-party computation schemes [52, 23], the use of one-way functions [22, 38], and dynamic accumulators [7, 4].

Section 4 outlines an algorithm for generating and maintaining negative databases. These operations need to be adjusted in order to produce "hard" negative databases. There is a large body of work regarding the issues and techniques involved in the generation of hard-to-solve $\mathcal{NP}$-complete problems [30, 29, 39, 34] and in particular of SAT instances [35, 9]. Much of this

work is focused on the case where instances are generated without paying attention to their specific solutions. Efforts concerned with the generation of hard instances when there is a specific solution we want the instance to possess include [18, 1]. Finally, the problem of learning a distribution, whether by evaluation or generation [31, 37], is also closely related to constructing the sorts of databases in which we are interested.

3 NEGATIVE DATABASES

The notion of negative databases was introduced in [15], whereby for a given set of fixed length strings, called the positive database DB (*self*), all the possible records or strings not in DB are represented i.e. $U - DB$ (*nonself*) where U denotes the universe of all possible strings of the same length defined over the same alphabet. It was shown that the size of the resulting negative database, denoted NDB, can be constructed to be polynomially related to the size of DB, even though ($U - DB \gg DB$) in the expected case. The intuition behind our compact representation is that there must be subsets of strings with significant overlaps, and that these overlaps can be used to represent these subsets succinctly. We have adopted the strategy of extending the alphabet over which strings are defined, in this case binary, to include an extra symbol $*$ known as the don't care symbol. A string exhibiting the $*$ at position i represents two strings; one with a 1 at position i and one with a 0 at position i with the remaining positions unchanged, as the following example illustrates. Position i in the string is referred to as a "defined position" if it contains either a 0 or a 1.

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

Including this third symbol allows one entry (or record) in the negative database to represent several strings (records) in $U - DB$*. A string x is represented in NDB —meaning x is not in DB, if there is at least one string y in NDB that matches it, otherwise the string is in DB. Two strings are said to match if all of their positions match; the don't care symbol is taken to match everything.

* We consider DB to remain defined over the $\{0,1\}$ alphabet.

Esponda *et al.* [15] give two algorithms for creating an NDB under the representation described above. One common feature is that both take as input DB—meaning DB must be held in its entirety at one time. Both operate by examining chosen subsets of bit positions to determine which patterns are not present in DB and must be depicted in NDB, the basic objective being to find the subsets of bit positions that serve to represent the largest number of strings in $U - DB$.

An interesting property of this representation concerns the difficulty of inferring DB given NDB. For an arbitrary set of strings defined over $\{0, 1, *\}$ determining which strings are not represented is an $\mathcal{NP}$-hard problem. To sustain this claim it is sufficient to note that there is a mapping from Boolean formulae to NDBs such that finding which entries are not represented by NDB (that is, which entries are in DB) is equivalent to finding satisfying assignments to the corresponding boolean formula, which is known to be an $\mathcal{NP}$-hard problem. The specifics of the proof can be found in Ref. [15], an example of the mapping is given in Figure 1.

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

FIGURE 1
Mapping SAT to NDB: In this example the boolean formula is written in conjunctive normal form and its 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 $\{x_1$= TRUE, x_2= TRUE, x_3= TRUE, x_4= FALSE, x_5= FALSE $\}$ corresponding to string 11100 is *not* represented in NDB.

4 CREATING AND MAINTAINING NEGATIVE DATABASES

In this section we present an on-line algorithm for creating and maintaining a negative database under the representation discussed in Section 3. Negative

databases should be viewed as logical containers of strings or detectors and it is important to point out that when the strings stored therein implement some partial matching rule, as is the case in AIS, removing or inserting a single string changes the definition of DB or *self* according to the particulars of that match rule.

The algorithms discussed in this section have the flexibility to create negative databases with varying structures (see instance-generation models [35, 9, 10]), an implementation of the algorithms must make some restrictions in order to yield NDBs that are hard to reverse on average. The following are some properties, regarding string matching, that the algorithms take advantage of:

Property 1: A string y is subsumed by string x 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.

Property 2: 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 one with those n positions set to the don't care symbol.

4.1 Initialization

A natural default initialization of DB is to the empty set. The corresponding initialization of NDB would be to U—the set of all strings. As discussed in Section 3 an NDB contains strings defined over $\{0, 1, *\}$ so one possible initial state for NDB would be simply to store the string $*^l$, where l is the string length. This clearly matches every string in U, but doing so would trivially defeat our purpose of making it difficult for someone to know exactly what NDB represents. We need to make it hard to know what DB is, even when DB is empty.

The algorithm presented in Figure 2 creates a database whose strings will match any string in U (see example in Fig. 3). The rationale behind it comes from the isomorphism between NDBs and Boolean formulae (Sect. 3, Fig. 1). Hence, our algorithm is designed to potentially create the equivalent of unsatisfiable SAT formulas of l variables. The high-level strategy is to select m bit positions and create, for each possible bit assignment V_p of these positions, a string with V_p and don't care symbols elsewhere. A negative database created in this way will have 2^m records, each matching 2^{l-m} distinct strings, clearly covering all of U.

Figure 2 modifies this strategy to expand the number of possible NDBs output by the algorithm. The modifications are:

- Potentially add more than one string to match a specific pattern (line 3).
- Augment the original pattern with n other positions. This allows each of the l possible positions to be specified in the resulting pattern. However the choice of n has great impact on the complexity of the algorithm as line 6 loops 2^n times. A value of 3 will suffice to generate some types of 3-SAT formulas (see Fig. 1) while keeping the complexity reasonable.
- Using a subset of positions of the selected patterns to create an entry (line 8–9).

It is straightforward to verify using the properties laid out at the beginning of the section that the algorithm produces an NDB that matches every string in U.

The purpose of the current choices of k_1 and k_2, lines 3 and 8 respectively, is to give the algorithm the necessary flexibility to generate genuinely hard-to-reverse NDB instances. We return to this question in Sections 5 and 6, as some restriction to these values might be warranted to produce negative databases with some desired structure.

Empty_DB_Create(l)
1. Pick $\lceil log(l) \rceil$ bit positions at random
2. for every possible assignment V_p of these positions{
3. select k_1 randomly $1 <= k_1 <= l$
4. for j=1 to k_1{
5. Randomly select $0 \leq n \leq O(log_2(l))$
6. select an additional n distinct positions
7. for every possible assignment V_q of these positions{
8. Pick k_2 bits at random from the patterns V_p and V_q
9. Create a entry for NDB with the k_2 chosen bits and fill the remaining $l - k_2$ positions with the don't care symbol.}}}

FIGURE 2
Empty_DB_Create. Randomly creates a negative database that represents every binary string of length l.

Empty_DB_**Create**(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 3
Possible states of NDB after successively performing initialization, deletion and insertion of a string.

4.2 Updates

We now turn our attention to modifying the negative database NDB in a dynamic scenario. The policies and algorithms used for selecting which strings should be added or retired remain application specific. It is worth emphasizing that the meaning 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 $U - DB$".

The core operation for both the insert and delete procedures, presented in Figure 4 (Negative_Pattern_Generate), takes a string x and the current NDB and outputs a string y that subsumes x without matching any other string in DB. The function starts by picking a random ordering π of the bit positions, to be used in all string operations, so as to remove biases from later choices. Lines 2–6 find a minimum subset of bits from the input string x such that no string outside $\{U - DB\} \cup \{x\}$ is matched. Step 4 of the algorithm ensures that inserting a don't care symbol at the selected position doesn't cause the string to match something in DB. Property 1, listed at the beginning of the section, establishes that the resulting string matches whichever strings the original input string x matched. Steps 7–9 create a string containing the

Negative_Pattern_Generate(NDB**,** x**)**

1. Create a random permutation π
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 Property 1 in Section 4.
[b] π' is the inverse permutation of π.

FIGURE 4
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.

pattern found in the previous steps plus possibly some extra bits; note that the added bits were part of the original input string x so it is automatically guaranteed that the result will subsume x. It is important to emphasize that, for an actual implementation of the algorithm, the value of t (line 7) might be restricted or even fixed to provide a desired NDB structure (see Section 5).

Insert into NDB

The purpose of the insert operation is to introduce a subset of strings into the negative database (thereby removing them from DB) while safeguarding its irreversibility properties. Figure 5 shows the pseudocode of the insert operation, lines 1 and 2 enable the procedure to create several entries in NDB portraying x, as for the initialization of NDB shown in Fig. 2, the actual number of entries should be set to accommodate efficiency constraints and to preserve the irreversibility of NDB. Likewise steps 3–5 set some of the unspecified positions of x (if any) so that it may be possible for a set of strings representing x, that exhibit bits not found in x, to be entered in NDB (see property 2 at the top of the section). Finally the call to Negative_Pattern_Generate (see Sect. 4.2 and Fig. 4) produces a string representing x which is then inserted

Insert(x**,** NDB**)**
1. Randomly choose $1 \leq j \leq O(l)$
2. for $k = 1$ to j do
3. Randomly select $0 \leq n \leq O(log_2(l))$
4. Randomly select from x, n distinct unspecified bit positions
5. for every possible bit assignment B_p of the selected positions
6. $x' \leftarrow x \cdot B_p$
7. $y \leftarrow$ Negative_Pattern_Generate(NDB, x')
8. add y to NDB

FIGURE 5
Insert x into NDB. Inserting a string into NDB amounts to removing the corresponding binary strings from DB.

in NDB (see example in Fig. 3).

Delete from NDB

This operation aims to remove a subset of strings from being represented in NDB. It is worth noting that this operation cannot simply be implemented by looking for a particular entry in NDB and removing it, since it may be the case that a string is represented by several entries in NDB and an entry in NDB can in turn represent several strings, some of which might not be our intent to remove. Figure 6 gives a general algorithm for removing a string or set of strings from being depicted in NDB, note that input x may be any string over $\{0, 1, *\}$ an thus many strings may cease from being represented by a single call.

The algorithm takes the current NDB and the string to be removed x as input. Line 1 identifies the subset, D_x, of NDB that matches x and removes it. As mentioned previously, removing an entry that matches x might also unintentionally delete some additional strings. Lines 3–7 reinsert all the strings represented by D_x except x. For each string y in D_x that has n unspecified positions (don't care symbols) there are n strings to be inserted into NDB that match everything y matches except x. Each new string y'_i is created by using the specified bits of y and the complement of the bit specified at the i^{th} position of x as the following example illustrates:

x	D_x	All but x
101001	1*1*0*	111*0* 1*110* 1*1*00

To see that this in fact excludes x from NDB, and nothing else, note the following: Each new string y_i', by construction, differs from x in its i^{th} position therefore none of the new strings match x. If a totally specified string $z \neq x$ is matched by $y \in D_x$ then z must have the same specified positions as y, now, since 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'. Finally, observe that since y subsumes each new entry y_i' no unwanted strings are included by the operation (see example in Fig. 3).

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

FIGURE 6
Delete from NDB. Delete a string from NDB amounts to appending the corresponding binary strings to DB.

One very important fact to point out about this algorithm is that it may cause the size of NDB to grow unreasonably. It is important for any implementation to prevent $|D_x|$, the number of entries in NDB that match a particular totally specified string, from being a function of the size of the negative database and/or to instrument a clean-up operation that bounds the size of NDB. The following section investigates the impact of this operation on the size of NDB and introduces a scheme that allows control over the growth of NDB.

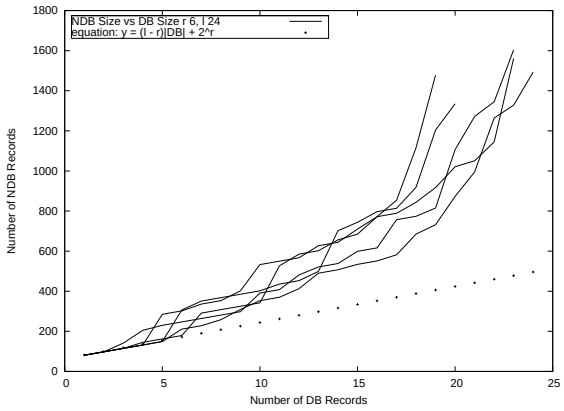

FIGURE 7
NDB size as a function of the number of consecutive Deletes (DB size) for strings of length $l = 24$ and $r = 6$ specified bits. NDB initially represents an empty DB. Strings used as arguments for Delete are selected randomly. Results of 5 independent runs are plotted, each run ends when $|NDB| \geq 3y$ where y represents the ideal NDB growth.

5 EXPERIMENTAL RESULTS

In this section we investigate how the size of a negative database, NDB, is affected by a sequence of Delete operations (Fig. 6). We focus our attention on Delete as it dominates the growth of NDB (see Section 4.2).

For the purpose of the experiments, a specific structure is imposed on NDB, one in which each entry has at least r specified positions. We fix the values of $k_1 = 1$ and $k_2 = r$ in lines 3 and 8 of the initialization algorithm (Fig. 2) and set the value of t in line 7 of Negative_Pattern_Generate (Fig. 4) to ensure that the resulting pattern V_k in line 9 has no less than r bits specified. These restrictions allow for a cleaner analysis of the impact of the Delete operation, and generate a class of instances that are expected to be harder to solve than those where an arbitrary number of specified bits per string is allowed (see Ref. [35]). Furthermore, we ensure that only one string is added to NDB per call to the Insert operation (Fig. 5) by setting the values of $j = 1$ and $n = 0$ in lines 1 and 4 respectively.

As was pointed out in Section 4.2 that the number of strings appended to NDB, after each call to Delete (Fig. 6), is a function of the number of

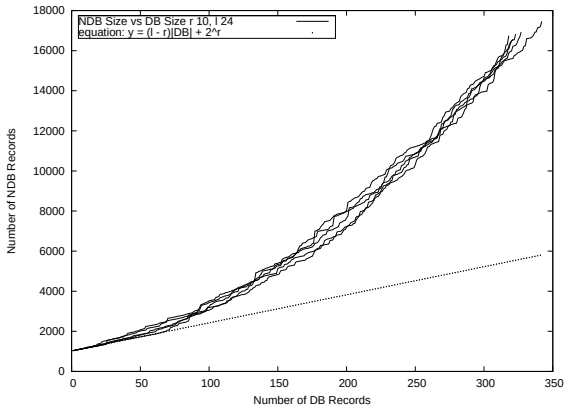

FIGURE 8
NDB size as a function of the number of consecutive Deletes (DB size) for strings of length $l = 24$ and $r = 10$ specified bits. NDB initially represents an empty DB. Deleted strings are selected randomly. Results of 5 independent runs are plotted, each run ends when $|NDB| \geq 3y$ where y represents the ideal NDB growth.

entries, D_x, that match the binary string x that is to be deleted, and of the number of specified bits each string in D_x has. For our current model, the number of strings added to NDB during a single call to Delete is simply $(l - r)|D_x|$. Figures 7 and 8 show the effects of executing a sequence of Delete operations on an NDB that initially represents every binary string of length l (initially stands for an empty DB) for two different values of r (note that executing a Delete operation is equivalent to inserting a string into DB). Line $y = (l-r)|DB| + 2^r$ depicts the ideal NDB growth; this occurs when each call to Delete causes the addition of at most $(l - r)$ strings to NDB i.e. $|D_x| = 1$, its plot is shown alongside the experimental results for comparison. Notice how the size of NDB stays close to y for an initial number of operations and quickly starts to deviate as the number of strings in DB surpasses some threshold. Figures 7 and 8 show the growth of NDB until its size has reached $3y$.

5.1 Controlled Growth

The results of the preceding section show the increasing growth rate of NDB after an initial linear progression. In this section we propose a scheme aimed

at reducing the growth rate of NDB so as to increase the number of strings that can be added to DB efficiently.

The operation presented in Figure 9 takes as input a negative database NDB and outputs a negative database NDB' that represents exactly the same set of binary strings i.e. 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. For the purpose of the current experiments τ is set to $2^{r-|K|}$ where r is the minimum number of specified bits and $|K|$ is the number of specified bits in pattern K.

Clean-up(NDB**)**
1. Randomly select a string x from NDB
2. Find a subpattern K of x not found in DB [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 Negative_Pattern_Generate (Fig.4.

FIGURE 9
Clean-up. Outputs a negative database that represents the same binary strings as its input NDB but with a different set of entries.

Lines 1–2 (Fig. 9) find a subpattern K of a string in NDB, such that no string in DB has that pattern i.e. every string in $\{0,1\}^l$ with such a pattern is represented in NDB (see Property 1 in Sect. 4). Line 3 finds all NDB entries D_K that exhibit this pattern and line 5 removes them. Note that, by removing D_K, only strings in $\{0,1\}^l$ that have K stop from being represented in NDB. Lines 6–7 reinsert every string and only strings with K as a subpattern into NDB (see Sect. 4.2).

The clean-up operation can be executed whenever spare processing time is available and it should be performed several times to increase the chances that the size of NDB will be significantly reduced. Figures 10 and 11 show the results of executing the clean-up operation interspersed with Delete oper-

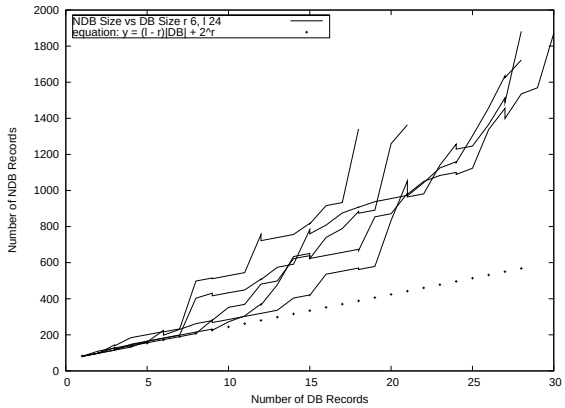

FIGURE 10
NDB size as a function of the number of Deletes (DB size) interspersed with Clean-up operations for strings of length $l = 24$ and $r = 6$ specified bits. The Clean-Up procedure is called 3000 consecutive times for each three consecutive Delete operations. NDB initially represents an empty DB. The strings used as argument for Delete are selected randomly. Results of 5 independent runs are plotted, each run ends when $|NDB| \geq 3y$ where y represents the ideal NDB growth.

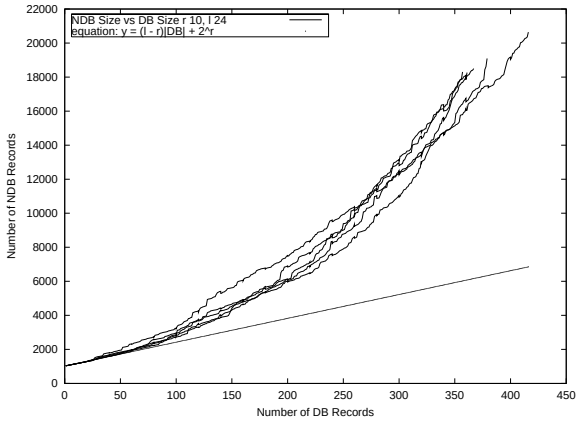

FIGURE 11
NDB size as a function of the number of Deletes (DB size) interspersed with Clean-up operations for strings of length $l = 24$ and $r = 10$ specified bits. The Clean-Up procedure is called 6000 consecutive times for each 20 consecutive Delete operations. NDB initially represents an empty DB. The strings used as argument for Delete are selected randomly. Results of 5 independent runs are plotted, each run ends when $|NDB| \geq 3y$ where y represents the ideal NDB growth.

ations. The plots show how the number of Delete operations is increased (on average roughly 16% for $r = 10$ and 9% for $r = 6$) as compared to figures 7 and 8.

On a final note, its relevant to point out that as the number of operations on NDB increases the incidence of strings with more than r specified bits may also grow. One possibility to deal with this phenomenon is to increase the minimum number of specified bits required per record, and reinsert all those negative entries with fewer specified bits. This will have the immediate effect of increasing the size of NDB, since increasing the specified bits in a record requires inserting additional strings to ensure DB is represented exactly (see Property 2 is Sect. 4) but will reduce the rate at which NDB grows with each update.

6 DISCUSSION

In this paper we have reviewed the concept of negative databases and introduced them as a means for storing strings or detector sets in the context of

anomaly detection systems based on string matching. Negative representations of data can provide an extra level of protection for systems in which acquiring the detector set (the set of strings that detect anomalies) might produce a security breach. We described an algorithm for generating negative databases on-line that, unlike the previous work where the positive database was assumed to exist at one place and at one time in order to obtain its negative representation, allows for the negative representation to be updated dynamically.

In applications of AIS to anomaly detection, the set of detectors or strings typically implement a partial matching rule. This allows the system to include, in the definition of *self*, strings that have not been observed before (also known as a generalization). This contrasts with the previous negative database work where the negative information of a set is represented exactly. An important observation in regards to partial matching is that, for the irreversibility result to hold, it must be the case that the match rule complies with the generalized satisfiability problem [21] according to the isomorphism with Boolean formulas described in Section 3. Moreover, even though it was shown in [15] that finding DB given only NDB is $\mathcal{NP}$-hard, this does not mean that every NDB is hard to reverse. The algorithms presented in Section 4 have a series of free parameters that will need to be tuned in order to realize the irreversibility properties afforded by the negative representation.

We have developed a preliminary version of the on-line algorithms presented here and those introduced in Ref. [15], referred to as the batch method. Quantitative results are still premature but some qualitative observations are relevant: Unlike the batch method, where the critical time cost is querying many negative patterns against the positive database, the online version spends its time querying the input record against the negative database. The negative database is typically larger than the positive database, and has been so in our tests.

The prototype, based on the algorithms in this paper is limited to records constructed from small two or three letter alphabets (16 to 24 bits). In order to evaluate the difficulty of retrieving positive records given only NDB we convert NDB into a Boolean formula, taking advantage of the relationship an NDB has with SAT, and input it to a well-known SAT solver [36, 2]. The solver returns the difficulty of obtaining a solution (specific to the particular heuristics used in the solver). One interesting observation is that the complexity of reversing the output of the on-line algorithm is significantly higher that that of the batch version. It appears that starting from an unsatisfiable formula and gradually adding satisfying assignments (adding records to DB,

deleting them from NDB) is more challenging for the heuristics employed by the solver. We also expect the difficulty of reversing a negative database to increase as the length of the strings in the database grow.

Both the processes of insertion and deletion cause NDB to grow in size. We provide some experimental results that show how the size of the negative database is affected by updates (Delete in particular (Fig. 6)), and introduce a clean-up procedure (Fig. 9)) whose aim is to reduce the size of NDB and thus control its growth. This operation transforms an NDB into another negative database that represents the same set of binary strings, in the future we would like to investigate how a variant of this operation may be used to search the space of hard-to-reverse negative databases.

Negative detection has been a trademark characteristic of artificial immune systems since they were first introduced and it is often lauded for its ability for distributed detection and its flexibility in detecting anomalies. Our research has led us to investigate the more general question of negative data representations and their properties. This led to the discovery that representing negative information in a certain manner exhibits an interesting and potentially useful property, namely that it makes it hard to recover the corresponding positive information. In the context of AIS for anomaly detection it adds an extra layer of security by making it hard to retrieve the profile of the system being monitored by simply analyzing the detector set. In other applications involving databases, it enhances privacy by naturally allowing only certain types of queries. Our current efforts are focused on the practical aspects of generating negative databases as well as in drawing out some additional properties that distinguish them from their positive counterpart.

7 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. also thanks CONACYT grant No. 116691/131686.

REFERENCES

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

[2] Boolean Satisfability Research Group at Princeton, (2004). zChaff. http://ee.princeton.edu/ chaff/zchaff.php.

[3] M. Ayara, J. Timmis, R. de Lemos, L. N. de Castro, and R. Duncan. (September 2002). Negative selection: How to generate detectors. In J. Timmis and P. J Bentley, editors, *Proceedings of ICARIS*, pages 89–98, Canterbury, UK. University of Kent at Canterbury Printing Unit.

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

[5] D. W. Bradley and A. M. Tyrrell. (12-14 July 2001). 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. IEEE Computer Society.

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

[7] J. Camenisch and A. Lysyanskaya. (2002). 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.

[8] D. L. Chao and S. Forrest. (2003). Generating biomorphs with an aesthetic immune system. In R. Standish, M. A. B., and H. A. Abbass, editors, *Proceedings of Artificial Life VIII*, pages 89–92, Cambridge, Massachusetts. MIT Press.

[9] S. A. Cook and D. G. Mitchell. (1997). 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.

[10] J. M. Crawford and L. D. Anton. (1993). Experimental results on the crossover point in satisfiability problems. In R. Fikes and W. Lehnert, editors, *Proceedings of AAAI*, pages 21–27, Menlo Park, California. American Association for Artificial Intelligence, AAAI Press.

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

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

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

[14] F. Esponda, S. Forrest, and P. Helman. (2003). The crossover closure and partial match detection. In J. Timmis, P. J. Bentley, and E. Hart, editors, *Proceedings of the ICARIS*, pages 249–260.

[15] F. Esponda, S. Forrest, and P. Helman. (2004). Enhancing privacy through negative representations of data. Technical report, Univerity of New Mexico.

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

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

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

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

[20] A. A. Freitas and J. Timmis. (2003). Revisiting the foundations of AIS: A problem oriented perspective. In J. Timmis, P. J. Bentley, and E. Hart, editors, *Proceedings of ICARIS*, pages 229–241.

[21] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company.

[22] O. Goldreich. (1997). On the foundations of modern cryptography. *Lecture Notes in Computer Science*, 1294:46–??

[23] S. Goldwasser. (1997). Multi party computations: past and present. In *Proceedings of PODC*, pages 1–6. ACM Press.

[24] F. Gonzalez, D. Dasgupta, and L. F. Nino. (2003). A randomized real valued negative selection algorithm. In J. Timmis, P. J. Bentley, and E. Hart, editors, *Proceedings of ICARIS*, pages 261–272.

[25] J. Greensmith and S. Cayzer. (2003). An AIS approach to semantic document classification. In J. Timmis, P. J. Bentley, and E. Hart, editors, *Proceedings of ICARIS*, pages 136–146.

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

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

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

[29] R. Impagliazzo, L. A. Levin, and M. Luby. (1989). Pseudo-random generation from one-way functions. In *Proceedings of STOC*, pages 12–24. ACM Press.

[30] R. Impagliazzo and M. Naor. (1989). Efficient cryptographic schemes provably as secure as subset sum. In IEEE, editor, *FOCS*, pages 236–241, Silver Spring, MD. IEEE Computer Society Press.

[31] M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie. (1994). On the learnability of discrete distributions. In *Proceedings of STOC*, pages 273–282. ACM Press.

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

[33] P. May, K. C. Mander, and J. Timmis. (Sep 2003). Software vaccination: An AIS approach. In J. Timmis, P. J. Bentley, and E. Hart, editors, *Proceedings of ICARIS*, pages 81–92, Edinburgh, UK. Springer-Verlag.

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

[35] D. Mitchell, B. Selman, and H. Levesque. (1992). Problem solving: Hardness and easiness - hard and easy distributions of SAT problems. In *Proceeding of AAAI-92*, pages 459–465. AAAI Press, Menlo Park, California, USA.

[36] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and Sh. Malik. (June 2001). Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*.

[37] M. Naor. (1996). Evaluation may be easier than generation (extended abstract). In *Proceedings of STOC*, pages 74–83. ACM Press.

[38] M. Naor and M. Yung. (1989). Universal one-way hash functions and their cryptographic applications. In *Proceedings of STOC*, pages 33–43, New York, NY 10036, USA. ACM Press.

[39] Odlyzko. (1991). The rise and fall of knapsack cryptosystems. In *Proceedings of PSAM*.

[40] G. J. Popek. (June 1974). Protection structures. *COMPUTER*, 7(6):22–33.

[41] J. H. Saltzer and M. D. Schroeder. (September 1975). The protection of information in computer systems. *Communications of the ACM*, 17(7)(9):1278–1308.

[42] S. Sathyanath and F. Sahin. (2001). Artificial immune systems approach to a real time color image classification problem. In *Proceedings of IEEE-SMC*.

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

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

[45] A. O. Tarakanov, V. A. Skormin, and S.P. Sokolova. (2003). *Immunocomputing:Principles and Applications*. Springer-Verlag.

[46] D. W. Taylor and D. W. Corne. (Sep 2003). An investigation of negative selection for fault detection in refrigeration systems. In J. Timmis, P. J. Bentley, and E. Hart, editors, *Proceedings of ICARIS*, pages 34–45, Edinburgh, UK. Springer-Verlag.

[47] P. A. Vargas, L. Nunes de Castro, R. Michelan, and F. J. Von Zuben. (Sep 2003). An immune learning classifier network for automated navigation. In J. Timmis, P. J. Bentley, and E. Hart, editors, *Proceedings of ICARIS*, pages 69–80, Edinburgh, UK. Springer-Verlag.

[48] P. Wayner. (2002). *Translucent Databases*. Flyzone Press.

[49] S. T. Wierzchon. (2000). 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. Physica-Verlag.

[50] S. T. Wierzchon. (2001). 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. Physica-Verlag.

[51] P. D. Williams, K. P. Anchor, J. L. Bebo, G. H. Gunsch, and G. D. Lamont. (2001). 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. Springer.

[52] A. Yao. (1982). Protocols for secure computation. In IEEE, editor, *FOCS*, pages 160–164, Silver Spring, MD, USA. IEEE Computer Society Press.