On-line Negative Databases

Fernando Esponda, Elena S. Ackley, Stephanie Forrest, and Paul Helman

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

Abstract. 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 described, which allow on-line creation and updates of negative databases, and 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 [25, 14, 13, 52, 53], success in distinguishing self from nonself [21, 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 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 Magritte 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 a 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 [20]. Anomaly-detection systems

based on these ideas can be found in [49, 3, 54, 33, 11, 27–29]. Other applications have also been proposed that range from image classification to recommender systems [34, 50, 26, 45, 49, 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, 48].

Protecting information stored in a databases from examination by unauthorized parties has been a concern since the very inception of the field [44, 43, 47]. Some approaches relevant to the current discussion involve cryptographic protection of databases [18, 46, 51], multi-party computation schemes [55, 24], the use of one-way functions [23, 39], 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 [31, 30, 40, 35] and in particular of SAT instances [36, 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 [19, 1]. Finally, the problem of learning a distribution, whether by evaluation or generation [32, 38], is also closely related to constructing the sort 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, analytically, 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 $U - DB$[1]. 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.

Esponda, Forrest and Helman [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], and 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

Fig. 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. In 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.

[1] We consider DB to remain defined over the {0,1} alphabet.

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 [36, 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. And, 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. These extra positions are necessary in order to allow strings to have distinct specified bit positions.

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 7–8).

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 7 respectively, is to give the algorithm the necessary flexibility to generate genuinely hard-to-reverse NDB instances. We return to this question in Section 5, as some restriction to these values might be warranted to produce hard instances in practice.

```
Empty_NDB_Create(l)
1. Pick ⌈log(l)⌉ bit positions at random
2. for every possible assignment V_p of this positions{
3.     select k_1 randomly 1 <= k_1 <= l
4.     for j=1 to k_1{
5.         select an additional n distinct positions
6.         for every possible assignment V_q of these positions{
7.             Pick k_2 bits at random from V_p · V_q.
8.             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.}}}
```

Fig. 2. Empty_NDB_Create. Randomly creates a negative database that represents every binary string of length l.

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, since what we are storing is a representation of what is *not* is some database DB, the meaning of the insert and delete operations are inverted from their traditional sense. For instance, the operation "insert x into DB" would have to be implemented as "delete x from NDB" and "delete x from DB" as "insert x into NDB".

The core operation for both the insert and delete procedures, presented in Figure 4, 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 so as to remove biases from

Empty_NDB_**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

Fig. 3. Possible states of NDB after successively performing initialization, deletion and insertion of a string.

later choices. Lines 2–6 find a minimum subset of bits from the input string x such that no string outside $\{U - DB\} \bigcup \{x\}$ 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 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.

Negative_Pattern_Generate(x**,** NDB**)**
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 flipped
4. if x' is subsumed by some string in $\pi(NDB)$
5. $\pi(x) \leftarrow \pi(x) - i^{th}$ bit (set value to $'*'$)
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 $\pi(x)$ and the bits indicated by R.
10. return $\pi'(V_k)$[a]

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

Fig. 4. Negative_Pattern_Generate. Take 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.

Insert into *NDB* The purpose of the insert operation is to introduce a subset of strings into the negative database 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 and 4 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 in NDB (see example in Fig. 3).

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

Fig. 5. Insert into NDB.

Delete 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 a entry that matches x might also unintentionally delete some additional strings. Lines 3–6 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. Create a new string y' using the specified bits of y and the complement of the bit specified at the i^{th} position of x.
6. Insert(y', NDB)

Fig. 6. Delete from NDB.

One very important fact to point out about this algorithm is that it may cause the size of NDB to grow unreasonably, even exponentially. It is important for any implementation to prevent the number of entries $|D_x|$ 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.

5 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 [22] 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 preliminary implementations of the algorithms presented here as well as that 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). We have made a number of restrictions to the algorithms as a first step in understanding its effects and to limit the size of NDB and its running time. We constrained the number of passes through the Insert algorithm Fig. 5 to one (in line 1 $j = 1$) and kept $k1 = 1$ and $n = 3$ for the initialization phase (Fig. 2 lines 3 and 5 respectively). Further, both the processes of insertion and deletion cause NDB to grow in size, so it will be indispensable in the future to implement a clean-up operation that eliminates redundant strings.

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 [37, 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 or 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. Finally, our current implementation of the on-line algorithm can produce strings with a variable or constant number of specified bits. This latter restriction is in accordance with [36] and has, in our experience, greatly increased the complexity of reversing NDB.

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, in this course we have found 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.

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. also thanks CONACYT grant No. 116691/131686.

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. Boolean Satisfability Research Group at Princeton. zChaff. http://ee.princeton.edu/ chaff/zchaff.php, 2004.
3. 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.
4. 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.
5. 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.
6. 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.
7. J. Camenisch and A. Lyasyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials.

8. 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.
9. 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.
10. 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.
11. D. Dasgupta and F. Gonzalez. An immunity-based technique to characterize intrusions in computer networks. *IEEE Transactions on Evolutionary Computation*, 6(3), June 2002.
12. L.N. de Castro and J.I. Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer-Verlag, 2002.
13. 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.
14. 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.
15. F. Esponda, S. Forrest, and P. Helman. Enhancing privacy through negative representations of data. Technical report, Univerity of New Mexico, 2004.
16. 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.
17. J. D. Farmer, N. H. Packard, and A. S. Perelson. The immune system, adaptation, and machine learning. *Physica D*, 22:187–204, 1986.
18. 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.
19. 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.
20. 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.
21. A. A. Freitas and J. Timmis. Revisiting the foundations of AIS: A problem oriented perspective. In Jonathan Timmis, Peter J. Bentley, and Emma Hart, editors, *Proceedings of the 2nd International Conference on Artificial Immune Systems (ICARIS)*, pages 229–241, Edinburgh, UK, Sep 2003. Springer-Verlag.
22. M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, San Francisco, 1978.
23. O. Goldreich. On the foundations of modern cryptography. *Lecture Notes in Computer Science*, 1294:46–??, 1997.

24. 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.
25. F. Gonzalez, D. Dasgupta, and L. F. Nino. A randomized real valued negative selection algorithm. In Jonathan Timmis, Peter J. Bentley, and Emma Hart, editors, *Proceedings of the 2nd International Conference on Artificial Immune Systems (ICARIS)*, pages 261–272, Edinburgh, UK, Sep 2003. Springer-Verlag.
26. J. Greensmith and S. Cayzer. An AIS approach to semantic document classification. In Jonathan Timmis, Peter J. Bentley, and Emma Hart, editors, *Proceedings of the 2nd International Conference on Artificial Immune Systems (ICARIS)*, pages 136–146, Edinburgh, UK, Sep 2003. Springer-Verlag.
27. S. Hofmeyr. *An immunological model of distributed detection and its application to computer security*. PhD thesis, University of New Mexico, Albuquerque, NM, 1999.
28. 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.
29. S. Hofmeyr and S. Forrest. Architecture for an artificial immune system. *Evolutionary Computation Journal*, 8(4):443–473, 2000.
30. 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.
31. 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.
32. M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie. On the learnability of discrete distributions. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 273–282. ACM Press, 1994.
33. 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.
34. P. May, K. C. Mander, and J. Timmis. Software vaccination: An AIS approach. In Jonathan Timmis, Peter J. Bentley, and Emma Hart, editors, *Proceedings of the 2nd International Conference on Artificial Immune Systems (ICARIS)*, pages 81–92, Edinburgh, UK, Sep 2003. Springer-Verlag.
35. R. C. Merkle and M. E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE-IT*, IT-24:525–530, 1978.
36. 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.
37. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and Sh. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, June 2001.
38. M. Naor. Evaluation may be easier than generation (extended abstract). In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 74–83. ACM Press, 1996.

39. 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.
40. Odlyzko. The rise and fall of knapsack cryptosystems. In *PSAM: Proceedings of the 42th Symposium in Applied Mathematics, American Mathematical Society*, 1991.
41. 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.
42. 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.
43. G. J. Popek. Protection structures. *COMPUTER*, 7(6):22–33, June 1974.
44. J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.
45. 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.
46. B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, Inc., New York, NY, USA, 1994.
47. A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts (Fourth Edition)*. Mc Graw Hill, 2002.
48. A. O. Tarakanov, V. A. Skormin, and S.P. Sokolova. *Immunocomputing:Principles and Applications*. Springer-Verlag, 2003.
49. D. W. Taylor and D. W. Corne. An investigation of negative selection 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. Springer-Verlag.
50. P. A. Vargas, L. Nunes de Castro, R. Michelan, and F. J. Von Zuben. An immune learning classifier network for automated navigation. In Jonathan Timmis, Peter J. Bentley, and Emma Hart, editors, *Proceedings of the 2nd International Conference on Artificial Immune Systems (ICARIS)*, pages 69–80, Edinburgh, UK, Sep 2003. Springer-Verlag.
51. P. Wayner. *Translucent Databases*. Flyzone Press, 2002.
52. 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.
53. 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.
54. 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.
55. 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.