# Defining Self: Positive and Negative Detection*

**Fernando Esponda**
**Stephanie Forrest**
Department of Computer Science
University of New Mexico
Albuquerque, NM 87131-1386
{fesponda,forrest}@cs.unm.edu

February 5, 2002

**Abstract**

In this paper we present two anomaly detection schemes based on the same partial matching rule. One focuses on the detection of valid patterns while the other on the detection of invalid patterns, termed positive and negative detection respectively. An analysis of both schemes provides a basis for their comparison, as well as a means to control the generalizations in which they incur.

## 1 Introduction

In [4], the negative-selection algorithm was introduced as a method for representing and generating distributed sets of detectors to perform change detection. The negative-selection algorithm was demonstrated and analyzed in the context of the *r-contiguous bits matching rule*—a partial-matching rule between two strings, similar to Hamming Distance. In that paper, a probabilistic expected-case analysis of the algorithm's performance was given, in the case where the protected data and the detectors are assumed to be static and the protected data are randomly generated. The analysis relied on the assumption that independently generated detectors would have independent detection capabilities. Use of the algorithm was illustrated with the problem of virus detection, and it was noted that observed performance on real data sets (for the problem of computer virus detection) was often significantly better than that predicted by the expected-case analysis. Presumably, this improved performance was due to regularities in the protected the data, known as "self." Although the negative-selection algorithm cannot match the performance of common check-sum and message-digest algorithms at detecting one-bit changes, it is noise tolerant, and the authors speculated that the algorithm might be useful in applications for which noise tolerance and distributed processing are important.

Ref [2, 3] reported a linear time algorithm for generating detectors (linear in the number of protected strings) which uses $O(2^r(l-r)^2)$ space, where $r$ is a threshold value and $l$ is the length of the strings. This new algorithm addressed an important limitation, as the cost of generating detectors using the original algorithm increased exponentially with the size of the protected data. Limits to the precision of coverage that is possible under a wide variety of matching rules were also identified. Gaps in the coverage arise because of the use of generalization, implemented as partial matching rules between pairs of strings. Ref [2] counted the number of such gaps, called *holes*, for certain partial matching rules and estimated a lower bound on the size of the detector set needed to achieve a given probability of detecting abnormal strings. Following up on this work, Wierzchon developed a low-space complexity algorithm for generating an optimal repertoire of detector strings [11, 10, 12], and included an analysis of the holes in coverage associated with his algorithm. In both these projects, it was assumed that both the protected data and the detectors are static.

*University of New Mexico Technical Report TR-CS-2002-02.

Lamont and Harmer applied negative selection to the problem of computer virus detection, focusing on different match rules [1]. And developed a prototype that demonstrates that this approach is both scalable and effective.

Hofmeyr [8, 6, 7] described a network intrusion detection system which incorporated a modified form of the negative-selection algorithm, together with several other mechanisms, including the use of permutation maps to achieve diversity of representation. His experiments were the first in this line of work to consider dynamically changing data sets (e.g., when the definition of normal behavior is either incomplete or non-stationary) and a distributed environment. The setting was network intrusion detection on a local area network, and the detectors monitored the flow of TCP SYN packets through the network. The original negative-selection algorithm was modified to accommodate dynamic data sets—detectors were generated asynchronously and allowed to undergo negative selection in the live environment of the network. Thus, the IDS featured rolling sets of detectors. Detectors generated at different times were potentially exposed to slightly differing patterns of self.

Kim and Bentley also implemented a network intrusion detection system which incorporates negative selection [9]. Their implementation used a significantly more complicated representation than [8], with a much larger alphabet, shorter strings, and more general matching thresholds (lower $r$ in $r$-contiguous bits matching). Like Hofmeyr, they use the original random-generation method of producing detectors. They report difficulties generating valid detectors and conclude that negative selection is infeasible on its own for realistic network intrusion detection. Williams et al. [13], however, report positive results when using negative selection on the network intrusion detection problem. Their results and approach more closely parallel that taken in Hofmeyr's work. Their implementation goes beyond [8, 9], however, in that it considers UDP and ICMP packets as well as TCP packets.

To summarize, a growing body of work is exploring the use of negative selection, using various matching rules, both on static and dynamically changing data sets. Some of this work focuses on developing better algorithms and some of this work focuses on example applications in realistic settings. Underlying all this work is the question of negative versus positive detection. That is, what are the relative merits of defining a set in terms of itself, the set of normal or allowable patterns, which we call positive detection, versus defining a set in terms of its complement, the set of abnormal or anomalous, patterns. This latter, we call negative detection.

On the face of it, negative detection might not seem like a promising candidate at all. For most realistic data sets, the set of positive patterns will be significantly smaller than its complement. Thus, it would make sense to derive a representation of the smaller set, rather than its enormous complement. There are several reasons why negative detection merits further study, however. First, it has been used successfully in several applications. Second, if we assume a closed universe then, from an information-theory perspective, both sets contain the same amount of information, which suggests that there might be equally compact representations of the negative patterns. A third point about negative detection is that it allows the detection process to be distributed across multiple locations with virtually no communication required among the distributed components. As the scale of anomaly detection problems increases, the issue of distributed detection may become more important. Indeed, the inspiration for the negative-detection approach came from the natural immune system, which uses negative detection in a massively distributed environment—the body.

To date, there has been little theoretical analysis of the relative merits of positive versus negative detection, and that is the goal of this paper. Such an analysis is complicated by different representations (including permutations), different matching rules, dynamically changing self sets, and dynamically changing detector sets.

In this paper, we first discuss the need for partial matching rules and generalization, as well as what this implies for comparing positive and negative detection. We then review the idea introduced in [8, 6, 7] of permutations as a way of compensating for over-generalization (or holes) in our coverage. We show that $r$-contiguous bits matching together with permutations reduces the existence of holes dramatically and how such a matching scheme compares to Hamming Distance. Next, we introduce a simplification of $r$-contiguous bits matching, called template matching, which simplifies the theoretical calculations and allows us, finally, to compare the complexity of positive with negative detection, under certain assumptions.

Throughout the paper, we assume that the detection problem is posed as a set $RS$(real-self) of strings $s$, all of fixed-length $l$ of which we can only access a sample $S$ at any given time. The universe of all $l$-length

strings is referred to as $U$, and the set of patterns to be detected is the set $U - RS$. We consider all strings to be binary although the analysis is easily generalizable for a larger alphabet. Finally, unlike the analysis first reported in [4], we study how many detectors are required to attain the maximum possible coverage of $U - S$. In many settings, the extent of the coverage might be profitably traded off against performance, simply by reducing the number of detectors in the system.

## 2 Partial Matching and Generalization

If we consider a static scenario in which all of the strings we want to classify, $RS$, are present at a given time, then a simple detection scheme, which we will refer to as *strict positive detection*, compares any new string against $RS$ to determine its membership. Alternatively, we can assess a string's membership in $RS$ by comparing it to its complement ($U - RS$), a method that we refer to as *strict negative detection*.

In many cases, however, it is infeasible to exactly specify $RS$, either because the specification is too large to enumerate exactly or because it is not known. In these cases, we need to distinguish $RS$ from $U - RS$ based on sampling. That is, we need to generalize from a sample of real self l we refer to as $S$, in effect, making a model of the sample. A common way of achieving such a generalization is through the use of pattern matching, or in our case of simple strings, partial matching rules. D'haeseleer referred to the patterns that are considered by the model as part of self, but that nevertheless have not been observed, as holes. It is the set $H$ containing all such strings that constitutes the model's generalization.

In order to make a fair comparison between positive and negative detection, we need to ensure that both methods are using the same generalization. Thus, it would be unfair to compare strict positive detection against a negative detection algorithm which uses generalization. We present such schemes in Sections 4 and 5.

In the work summarized in Section 1 negative detection was combined with the $r$-contiguous bits matching rule: Given a set $S$ of length $l$ strings to be to classified, a set of detectors $D$ also of length $l$ are generated in such a way that no detector in $D$ matches a string in $S$ at $r$ contiguous bit positions. A given string of length $l$ has $t = l - r + 1$ $r$-length windows, each of which is a possible region to satisfy the matching criterion. Any valid detector (one that matches none of the strings in $S$) will be composed of $t$ windows, each of which is not to be found in $S$. We choose the $r$-contiguous bits matching rule as a point of departure for this work because, much of the earlier research discussed above relies on it. Further, when $r$-contiguous bits is augmented with permutation masks, it is at least as powerful as the obvious alternative, Hamming Distance, as shown in section 3. The generalizations induced by $r$-contiguous bits, and any other partial matching rule, are sometimes beneficial and sometimes deleterious. In the case where the generalization reduces false positives, we can view it as a benefit, and in the case where it prevents detection of true positives it is a liability. Either way, however, we would like to understand the nature of these generalizations, and we would like to have control over them. As it turns out, a slight modification of $r$-contiguous bits, called template matching, improved our ability to control generalizations. There are two sources of holes that constitute the generalization $H$ for the $r$-contiguous bits matching rule. The first can be characterized as all strings in $U - S$ that constitute crossovers of strings from $S$ and are not in $S$ themselves. A *crossover* is defined between two adjacent windows that match in their $r - 1$ common positions. Two windows $w1 = v_i..v_{i+r-1}$ and $w2 = u_{i+1}..u_{i+r}$ crossover iff $[v_j = u_j \ \forall_j : i + 1 \leq j \leq i + r - 1]$. In other words, any string which has each of its $t$ windows matched by some window in $S$ is considered to be a self string.

However, these are not the only strings for which no detector can be generated. There is a second source of holes in which a string that is not comprised entirely of crossovers of $S$ is nevertheless treated as part of self, and is therefore undetectable. These holes have at least $r$ contiguous bits that are not matched by any of the corresponding windows in $S$. Thus, they have at least one distinctive window which could potentially be matched by a detector. However, in some cases there may not be any such available detector. This is because a detector of length $l$ must specify a pattern, that is not is $S$, in each of its windows. This combination could preclude a valid detector from being formed, as the following example illustrates:

> Let $l = 3$, $r = 2$, $S = \{011, 010\}$ and let $h = 110$ be one such hole, clearly a detector for $h$ cannot end with the pattern $*10$ since this is present is $S$, therefore it must start with $11*$ in

order to match $h$, but any string of length 3 whose first window is 11 must have as its second window either 10 or 11, in this case both patterns are contained in $S$, so a detector for $h$ cannot be generated.

# 3 Permutations and Diversity

If $H$, the set of holes, is too large, accurate discrimination between $S$ and $U-S$ will be unachievable. Hofmeyr introduced the concept of *permutation masks* as a way to control the size of $H$. A permutation mask is a reversible mapping that specifies a reordering of bits for all strings in $U$. In his network intrusion detection data set, he reported an improvement in detection ability by about a factor of three, when permutation masks were used.[1] One further benefit of this transformation is that if we consider all permutation masks, the $r$-contiguous bit matching rule is able to detect at least as many strings as Hamming Distance, as shown below. By controlling the number of permutations used, he was able to achieve greater control over the $r$-contiguous bits generalization. The proof follows:
Let:

- $x$,$d$ be strings of length $l$
- $R(x,d) : Strings \times Strings \rightarrow \mathbb{N}$ be a function that returns the length of the longest run of contiguous bits that match between strings $x$ and $d$. String $d$ is said to match string $x$ if $R(x,d) \geq r$.
- $H(x,d) : Strings \times Strings \rightarrow \mathbb{N}$ be a function that returns the Hamming distance between strings $x$ and $d$. String $x$ is matched by a string $d$ using the Hamming distance rule if $d$ and $x$ differ in at most $l-r$ bit positions.
- $\pi_i(x)$ denotes the $ith$ permutation of string $x$.
- $W_d$ denotes the set of all strings of length $l$, such that $x \in W_d \leftrightarrow R(\pi_i(x), \pi_i(d)) \geq r$ for some $\pi_i$ and a given string $d$. $W_d$ denotes all the strings that match $d$ under some permutation.
- $V_d$ be the set of all strings of length $l$ such that $x \in V_d \leftrightarrow H(x,d) \leq l-r$ for a given string $d$.
  Claim $V_d \subseteq W_d$
- If $x \in V_d$ then $H(x,d) \leq l-r$, so $x$ and $d$ have at least $r$ bits in common. We construct a permutation $\pi_j$ in the following way: Label the bits that differ between $x$ and $d$ as $c_1 c_2 ... c_n$, $1 \leq n \leq H(x,d)$ and the bits that match as $c_{n+1} c_{n+2} ... c_m$, $n+1 \leq m \leq l$. We define $\pi_j$ as $c_1 c_2 ... c_n c_{n+1} ... c_l$. Clearly $\pi_j$ is a valid permutation and $R(\pi_j(x), \pi_j(d)) \geq r$. Furthermore, if $d$ exists then $\forall_s : s \in S$, $H(s,d) > l-r$ and applying any permutation $\pi$ to every string $y$ in $S$ yields $R(\pi(s), \pi(d)) < r$ thus $\pi_j(d)$ is a valid detector for the $r$-contiguous bit matching rule.
- If $x \in W_d$ then there exists a permutation $\pi_i$ such that $R(\pi_i(x), \pi_i(d)) \geq r$, therefore $\pi_i(x)$ and $\pi_i(d)$ have at most $l-r$ different bits and $H(x,d) \leq l-r$. Nevertheless it is not necessarily the case that $d$ is a valid detector under the Hamming distance matching rule as the following example illustrates:
  Let $S = \{011, 110, 100\}$, $l$=3, $r$=2, $h$=000. Under the $r$-contiguous bits matching rule, we can generate a valid detector $d$=001 to match $h$, whereas under Hamming distance the potential detectors for $h$ are $000, 001, 010, 100$ all of which have a Hamming distance less that two with $h$, but also with all strings in $S$. Therefore $h$ is a hole under this rule but not under $r$-contiguous bits with permutation masks[5].

We conclude that $V_d \subseteq W_d$ and therefore that Hamming distance exhibits at least as many holes as the $r$-contiguous bit matching rule augmented with permutation masks. Operationally, for this scheme to achieve comparable coverage as the one provided by Hamming distance would require multiple sets of detectors $D$, each with its own transformation. Hence, this mechanism is amenable to a detection scheme that aims to be distributed and provides a means for controlling the amount of holes.

To illustrate how the permutation masks can in fact get rid of some holes consider the self strings $s1 = 101$ and $s2 = 000$ with $l = 3$ and $r = 2$. There are two crossover strings $h1$ and $h2$, for which no detector can be generated since neither string has a distinctive window that separates it from self:

[1] Hofmeyr's data was collected using a variant of pure permutations which was computationally more efficient.

| | | | | | | |
|---|---|---|---|---|---|---|
| Self | 010 | 001 | 100 | 100 | 001 | 010 |
| | 011 | 011 | 101 | 110 | 101 | 110 |
| | 100 | 100 | 010 | 001 | 010 | 001 |
| | 101 | 110 | 011 | 011 | 110 | 101 |
| $h$ | 110 | 101 | 110 | 101 | 011 | 011 |
| detector | 11* | 10* | 11* | 10* | 01* | 01* |
| templates | *10 | *01 | *10 | *01 | *11 | *11 |

Table 1: Undetectable strings under every permutation, for a given set $S$ and the $r$-contiguous bit matching rule.

$s1=$ [1][0][1] $\quad h1=$ [1][0][0]
$s2=$ [0][0][0] $\quad h2=$ [0][0][1]

If we apply the permutation by which the third bit position is next to the first bit, as the following picture illustrates. Then we can generate two detectors $d1$ and $d2$ that will be able to match $h1$ and $h2$ respectively.

$\pi(s1)=$ [1][1][0] $\quad \pi(h1)=$ [1][0][0] $\quad d1=$ [1][0][1]
$\pi(s2)=$ [0][0][0] $\quad \pi(h2)=$ [0][1][0] $\quad d2=$ [0][1][1]

### 3.1 Completeness

We can now ask whether or not coverage of $U - S$ is complete under $r$-contiguous bits matching with permutations. That is, are there still strings not in $S$ which go undetected if we use every possible permutation mask? In other words, is the $r$-contiguous bits matching rule with permutation masks equivalent, in the limit, to strict positive detection? If it is, then we are in a position to compare positive and negative detection directly. Unfortunately, the answer to our question is no—there are still some holes even if we use all possible permutations, in fact, every partial matching rule will always exhibit holes.

We show that there exist holes under $r$-contiguous bits matching with permutations by construction: Let $l$=3, $r$=2 and $S$={010,011,100,101} and consider the hole $h$=110. Table 1 lists all the possible permutations of $S$ and the substrings out of which a detector could be constructed for $h$ under each permutation.

As can be seen from table 1, under each permutation there are no available templates (that is, templates that don't match some string in $S$), and thus, it is impossible to generate a detector for $h$. Although we can construct such examples, it is difficult to characterize the set of all such holes under permutation.

## 4 Template Matching

An alternative to the $r$-contiguous bit matching rule in which detectors must be of length $l$, is to generate detectors of length $r$ together with the specification of the window to which they refer. This can also be thought of as generating detectors with wild cards, or don't-cares, each with exactly $r$ contiguous specified positions. In this way, the set of detectors $D$ will be comprised of strings of the form $v_1...v_r * *... * *$, $*u_2...u_{r+1} * ... * *$, etc. A string $s$ is said to be detected by a detector $d$ if $d$ matches $s$ in all of its specified positions. Lifting the restriction of having detectors be of length $l$ enables a template detector to match the non-self string $h$ from the previous example (table 1), limiting the holes of this scheme to be only those patterns that are in fact crossovers of strings in $S$.

However, even if every permutation mask is utilized there may still exist strings in $U - S$ which remain undetectable with templates. Consider the following scenario: let $l$=3, $r$=2, $S$={111,010,100} and $h$=110. Table 2 lists all 6 permutations of $S$ and $h$ and the possible template detectors needed to match $h$. In general a string $h$ cannot be detected under any permutation if all of its template combinations are in self. The template combinations of a string are all the templates, i.e. strings with unspecified bits, that result from

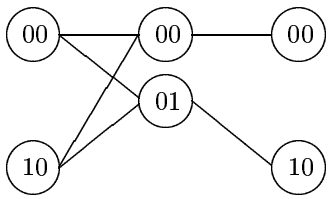
| Self | 111 | 111 | 111 | 111 | 111 | 111 |
|---|---|---|---|---|---|---|
| | 010 | 100 | 010 | 100 | 001 | 001 |
| | 100 | 010 | 001 | 001 | 100 | 010 |
| $h$ | 110 | 110 | 011 | 101 | 101 | 011 |
| detector | 11* | 11* | 01* | 10* | 10* | 01* |
| templates | *10 | *10 | *11 | *01 | *01 | *11 |

Table 2: Undetectable strings under every permutation, for a given set $S$ and the template matching rule.

all the possible ways of fixing $r$ bits. It is easy to see that if one such template does not exist in $S$ then there exists a permutation of the bit positions, such that these $r$ bits are contiguous and a template detector can be generated for $h$.

# 5 Positive Template Detection

One useful way to visualize the generalization induced by the template detectors, for a given permutation mask, is to construct a directed acyclic graph (DAG) $G = (V, E)$ where $|V|$ is the number of distinct bit patterns of length $r$ for each window of strings in $S$. Vertices are labelled by a 2-tuple $(i, j)$, where $i$ represents the window number and $j$ stands for the bit pattern. $|E|$ is the number of edges, an edge $e((i,j),(i',j')) \in E$ iff $i' = i + 1$ and the last $r-1$ bits of pattern $j$ match the first $r-1$ bits of pattern $j'$ where $(i,j),(i',j) \in V$. Take for instance a self set $S$ comprised of the following two strings $S = \{0000, 1010\}$ with $l = 4$, $r = 2$ then our graph $G = (V, E)$ will look as follows with $V = \{(1,00),(1,10),(2,00),(2,01),(3,00),(3,10)\}$ and $E = \{((1,00),(2,00)),\ ((1,00),(2,01)),\ ((1,10),(2,01)),\ ((1,10),(2,00)),\ ((2,00),(3,00)),\ ((2,01),(3,10))\}$. A picture of this follows, omitting the window number from the vertex labels.




One straightforward way to determine the number of holes is to find all possible paths in this graph starting at nodes in the first window. There are several efficient algorithms for this task. Here we present a mapping of the problem into the linear algebra domain where its set of tools might be of use. For instance, we can construct $l - r$ $2^r \times 2^r$ adjacency matrices representing the edges between consecutive levels and multiply them[2]. For the example given above, we would have the following matrices:

$$M_1 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \ M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$M_1 M_2 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

[2]Note that the matrices are extremely sparse, with at most 2 entries per row

The number of holes is then $CM_1M_2C^T - |S| = 2$. Where $C$ is a $1 \times 2^r$ vector with $C[i] = 1$ for all $1 \leq i \leq 2^r$.

This representation suggests an alternative form of detection. It can be thought of as generalized positive detection (generalized in the sense that not only strings in $S$ are considered as self) in which a string represents a route in the graph and is accepted only if it reaches an end node and is rejected otherwise. We will refer to each node in the graph as a positive detector, since determining if a string belongs to self requires it to be matched by the bit patterns representing the nodes in the graph. One straightforward implementation of this would have $t = l - r + 1$ (the number of windows in a string of length $l$) sorted lists of positive detectors, representing the distinct bit patterns that are present in self per window. This scheme captures the same set of strings as negative template detection with a complete set of detectors for a given $l$ and $r$.

# 6 Complexity Analysis

In this section, we consider the cost of positive versus negative selection, using template detectors. The relevant costs include the number of detectors required by each scheme, the time taken to determine if a string belongs to self or not, and the cost of insertion and deletion of patterns from each scheme.

## 6.1 Number of Detectors

Given a particular set $S$ it is straightforward to compute exactly how many detectors can be generated, both for the positive and negative detection schemes. In the positive case, it requires counting the number of distinct patterns for each of the $t$ windows that comprise the strings in $S$, whereas in the negative scheme, enumerating the distinct patterns that are not present in each window will result in the number of possible detectors. To provide an estimate of the average number of detectors needed to detect all patterns in $U - \{S \cup H\}$ as a function of the size of $|S|$, we note the following:

- The number of strings with a specific pattern in any given window of size $r$ is $2^{l-r}$.
- The probability of selecting at random one such string is $\frac{2^{l-r}}{2^l} = 2^{-r}$.
- Assuming $r$ is small compared to $l$, we approximate the probability that a randomly generated self set of $|S|$ unique strings will contain a specific pattern by $1 - \binom{|S|}{0}(2^{-r})^0(1-2^{-r})^{|S|} = 1 - (1-2^{-r})^{|S|}$.
- Following the approximation of the previous item, the expected number of distinct patterns, for a given window, is given by $2^r - 2^r(1-2^{-r})^{|S|}$.

The following equation denotes the average number of nodes or positive detectors that will be needed to characterize a self set as a function of its size.

$$E_p(|D|) = t(2^r - 2^r(1-2^{-r})^{|S|}) \tag{1}$$

Conversely, the expected number of detectors for our negative detection scheme is given by:

$$E_n(|D|) = t(2^r(1-2^{-r})^{|S|}) \tag{2}$$

To establish when one scheme provides a smaller set of detectors that the other, we determine the number of strings in $S$ for which both schemes yield the same size detector set $D$ (see figure 1). For this purpose we compute the intersection between (1),(2):

$$E_p(|D|) = E_n(|D|) = t(2^r(1-2^{-r})^{|S|}) = t(2^r - 2^r(1-2^{-r})^{|S|})$$

solving for $|S|$

$$|S| = \frac{log 2^{-1}}{log(1-2^{-r})} \tag{3}$$

Taking the following limits we can conclude that this intersection lies somewhere between $2^{r-1}$ and $2^r$.

$$\lim_{r\to\infty} 2^{r-1} - \frac{log 2^{-1}}{log(1-2^{-r})} = -\infty$$
$$\lim_{r\to\infty} 2^{r} - \frac{log 2^{-1}}{log(1-2^{-r})} = \infty$$

In a worst case scenario, the positive detection scheme may require $t2^r$ detection nodes when $|S| \geq 2^r$. Similarly, the negative detection scheme can also yield up to $t2^r$ detectors but only when there are no self strings whatsoever.

On the other hand, the number of detectors needed for negative detection can actually be smaller, since significant redundancy amongst them may be present. By a symmetric argument number the $t$ windows from left to right. Consider the following:

- Regardless of the composition of $S$ a detector must be generated for each pattern in the first window that is not present is $S$.
- For every window of size $r$, starting from the second window, and for every pair of patterns $w1 = v_i...v_{r+i-2}a$, $w2 = v_i...v_{r+i-2}\bar{a}$ in such a window:
  - If both $w1$ and $w2$ are present in self then we cannot generate any detector for them.
  - If neither is present then we need not generate detectors for them since the preceding window will be missing its prefix i.e. $bv_i..v_{r+i-2}$ or $\bar{b}v_i..v_{r+i-2}$ and the detectors for the previous window will also match strings with such a pattern.
  - If only one is present, say $v_i...v_{r+i-2}a$, then we must generate detector $v_i...v_{r+i-2}\bar{a}$ since no string in $S$ contains it.

With this in mind, the average number of detectors needed in the minimal set is given by:
Let $y = 2^r - 2^r(1-2^{-r})^{|S|}$ the expected number of distinct patterns for a window of size $r$ and $y_2 = 2^{r-1} - 2^{r-1}(1-2^{-r+1})^{|S|}$ the expected number of distinct patterns for a window of size $r-1$.

$$E_{min}(|D|) = 2^r - y + (l-r)(y - 2(y-y_2)) \tag{4}$$

Following the previous rationale we can also set an upper bound on the number of detectors required in the minimal set. The maximum number of self strings we can have without creating crossovers, thereby reducing the number of detectors, will exhibit only one of every pair of patterns $w1 = v_i...v_{r+i-2}a$, $w2 = v_i...v_{r+i-2}\bar{a}$ for each window. This results in a $t2^{r-1}$ detector set for a maximum of $2^{r-1}$ distinct self strings. Figure 1 shows the plots of the average number of detectors for the positive scheme, as well as the estimated number of the detectors for the negative scheme considering both the maximal and minimal sets.

## 6.2 Time and Space

Given a set $S$, one straightforward way of generating detectors will require determining for each window, what patterns are present. In the case of positive detection this will take time proportional to $|S|$ and the number $t$ of windows. If we consider sorting $S$ with respect to the numerical values of the patterns in each window, the time to generate the representation will be $O(t|S|^2 log|S|)$. The negative detection scheme will require determining which patterns are not is $S$ but there are only $2^r$ possible distinct patterns per window, since $|S|$ can be larger than $2^r$ and we must scan through the entire set, the time required is also $O(t|S|^2 log|S|)$. Determining if a string $s$ is part of self or not, will require searching for each of the $t$ patterns present is $s$, taking $O(tlog2^r) = O(tr)$ time for both representations as would insertion and deletion of detectors.

Space, on the other hand, can become a major objection because $O(tr2^r)$ storage space might be required. However, if $r$ is not too large this might manageable. For example, if we consider the parameters used by Hofmeyr [8] ($l = 49, r = 11$), the space required would be around one million bits or $130k$ bytes if all possible detectors are to be generated.

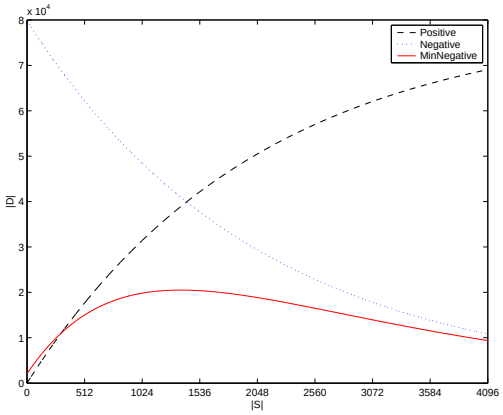



Figure 1: Average growth in the number of detectors for each scheme, as a function of $|S|$. Based on equations 1, 2, 4 with $l$=49 $r$=11.

# 7 Discussion

It is difficult if not impossible to state what is the best match rule for anomaly detection, since this is a domain specific issue. Nevertheless it is our belief that a greater understanding of a given rule allows us to exert more precise control over its performance, balancing its benefits and shortcomings. In this paper we presented a rule inspired by the $r$-contiguous bits match rule, that allows for both positive and negative detection schemes that are equivalent in terms of what they detect i.e. exhibit the same generalization. Furthermore, we've shown that this match rule exhibits a reduced number of holes when compared with its antecessor, and due to its simplified nature allows us to better characterize what precisely the holes are.

It is important to stress that we consider the existence of holes beneficial, since they constitute a prediction of what real self looks like, nevertheless, since the number of holes can grow quite large, understanding of this generalization and the means to control it is critical. Currently we are investigating how permutation masks affect the representation of $S$ in terms of how closely $S \cup H$ resemble real self ($RS$). This work has considered self to be a static entity, but we are ultimately interested in the behavior of this rule when self is in flux, and new samples become available over time. The results presented in this paper provide a grounding for such an analysis. We consider that the negative detection scheme is more amenable to a distributed implementation since distributing negative detectors is a straightforward task, while it is not obvious how to distribute positive detectors in such a way that the generalization is preserved. Finally, we've provided some guidelines in terms of the necessary number of detectors that enable us to choose between the proposed positive and negative schemes based upon the expected size of self.

# 8 Conclusion

In this paper we introduced a positive and a negative detection scheme for which the set of strings to be considered as part of self is the same. We presented an analysis that allows us to estimate the number of detectors each scheme will require as a function of the size of the available sample and provides a criteria for choosing one scheme over the other. Two straightforward implementations are put forth that hint to the time requirements for both schemes and their dynamic behavior. Finally we discussed a means of controlling this generalization and point out some of its limitations. In addition, we introduced a novel matching rule which is finer-grained than $r$-contiguous bits and thus has fewer holes.

## Acknowledgments

The authors gratefully acknowledge the support of the National Science Foundation (grants CDA-9503064, and ANIR-9986555), the Office of Naval Research (grant N00014-99-1-0417), Defense Advanced Projects Agency (grant AGR F30602-00-2-0584), the Intel Corporation, and the Santa Fe Institute. F.E. thanks Consejo Nacional de Ciencia y Tecnología (México) grant No. 116691/131686 for financial support. We would also like to thank the adaptive systems group and in particular Paul Helman, Justin Balthrop and Matthew Glickman.

## References

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

[2] 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.

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

[4] 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.

[5] Paul Helman. Example provided through personal communication.

[6] 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.

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

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

[9] 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.

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

[11] 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. Pysical-Verlag.

[12] 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 Paradigm in Soft Computing*, pages 438–458, Heidelberg New York, 2001. Physica-Verlag.

[13] 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, *rth International Symposium, Recent Advances in Intrusion Detection*, pages 117–133, Berlin, 2001. Springer.