A Formal Framework for Positive and Negative Detection Schemes

Fernando Esponda, Stephanie Forrest, and Paul Helman

***Abstract*— In anomaly detection, the normal behavior of a process is characterized by a model, and deviations from the model are called anomalies. In behavior-based approaches to anomaly detection, the model of normal behavior is constructed from an observed sample of normally occurring patterns. Models of normal behavior can represent either the set of allowed patterns (positive detection) or the set of anomalous patterns (negative detection). A formal framework is given for analyzing the tradeoffs between positive and negative detection schemes in terms of the number of detectors needed to maximize coverage. For realistically sized problems, the universe of possible patterns is too large to represent exactly (in either the positive or negative scheme). Partial matching rules generalize the set of allowable (or unallowable) patterns, and the choice of matching rule affects the tradeoff between positive and negative detection. A new match rule is introduced, called *r-chunks*, and the generalizations induced by different partial matching rules are characterized in terms of the *crossover closure*. Permutations of the representation can be used to achieve more precise discrimination between normal and anomalous patterns. Quantitative results are given for the recognition ability of contiguous-bits matching together with permutations.**

***Index Terms*— Intrusion detection, positive detection, negative detection, framework.**

l	Length of a string, a pattern or a packet. Typically given in bits.
r	Size of the sliding window.
w	A window, a specification of r adjacent symbol positions in a string. When the window length r is understood, it suffices for w to specify only a start position.
t	Number of windows in a string of length l.
U	All possible strings of length l for a given alphabet.
S	A subset of U used to denote a sample of "normal" strings.
RS	A subset of U used to denote the complete set of "normal" strings.
$x(t)$	A string or packet generated at time t.
G	A legitimate packet generating process.
B	A malicious packet generating process.
$y(t)$	A random variable indicating whether G or B generated $x(t)$.
$\mathcal{H}$	A hidden process governing the generation of packet $x(t)$.
$sample(t)$	A dynamic sample of G at instant t.
d	A string detector.
Υ	A set of detectors.
$CC_r(S)$	The crossover closure generation rule of S for a given window size r.
$x[w]$	For $x \in U$, $x[w]$ is x projected onto window w.
dMx	d matches x under match rule M.
$Scheme_{ND}(\Upsilon)$	The set of strings x in U such that $(\forall windows\ w)(\nexists d \in \Upsilon)(dMx)$.
$Scheme_{PD}(\Upsilon)$	The set of strings x in U such that $(\exists window\ w)(\exists d \in \Upsilon)(dMx)$.
$Scheme_{PC}(\Upsilon)$	The set of strings x in U such that $(\forall windows\ w)(\exists d \in \Upsilon)(dMx)$.
$Scheme_{NC}(\Upsilon)$	The set of strings x in U such that $(\exists window\ w)(\nexists d \in \Upsilon)(dMx)$.

Manuscript received July 17, 2002; revised January 10, 2003; accepted February 26 2003. In press: IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics.

I. Introduction

In anomaly detection, the normal behavior of a process is characterized by a model, and deviations from the model are called anomalies. In behavior-based approaches to anomaly detection, the model of normal behavior is constructed from an observed sample of normally occurring patterns. Models of normal behavior can represent either the set of allowed instances (positive detection) or the set of all anomalous instances (negative detection).

For most real problems, the set of positive instances is much smaller than the set of complementary ones. Thus, it would seem wise to derive a representation of the smaller set, rather than its enormous complement. However, there are several reasons why negative detection merits further consideration. First, it has been used successfully both in engineering applications and by naturally occurring biological systems. Second, if we assume a closed world, then from an information-theory perspective, the normal and abnormal sets both contain the same amount of information, which suggests that there might be equally compact representations of the negative patterns [1]. 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. This property allows several forms of distributed processing: checking small sections of a large object independently, independent detector sets (e.g., each one running on a separate machine), or evaluating each individual detector in a detector set independently. As the scale of anomaly detection problems increases, the need for distributed processing is likely to grow. Indeed, the original inspiration for the negative-detection approach came from the natural immune system, which uses negative detection in a massively distributed environment—the body.

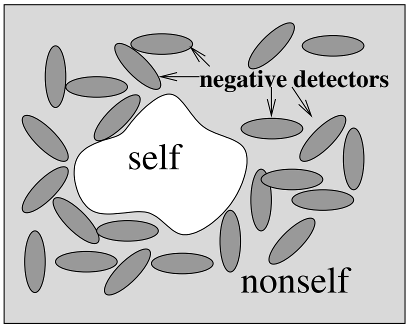


Fig. 1. Self Nonself Discrimination: A universe of data points is partitioned into two sets– self and nonself. Negative detectors cover subsets of nonself.

The negative-detection approach to anomaly detection, originally published as the negative-selection algorithm [2], [3], was modeled after a method used by the natural immune system to prevent autoimmunity [4]. In the immune system, certain cells known as T cells undergo a multi-stage maturation process in an isolated environment, an organ called the thymus. While in the thymus, T cells are censored against the normally occurring peptide patterns of the body, called *self*. T cells that react with self are deleted in the thymus before they can become active and cause autoimmunity. The only T cells allowed to mature and leave the thymus are those that survive this censoring operation[1]. Such cells then circulate through the body freely and independently, eliminating any material which they can bind. Because of the censoring process, such material is implicitly assumed to be foreign and is known as *nonself*. The translation of this mechanism into an algorithm for computers is straightforward. First, we assume that the anomaly-detection problem is posed as a set RS (real-self) of strings s, all of fixed-length l of which we can access only a sample S at any given time. The universe of all l-length strings is referred to as U, and the set of anomalous patterns to be detected is the set $U - RS$. We consider all strings to be binary although the analysis is generalizable to a larger alphabet. Candidate detectors are generated randomly and are censored against S; those which fail to match any of the strings in S are retained as active detectors. Such detectors are known as *negative detectors*, and if S is a good sample of RS each negative detector will cover (match) a subset of nonself. The idea is that by generating sufficient numbers of independent detectors, good coverage of the nonself set will be obtained. Figure 1 shows the relationship of these sets pictorially.

This algorithm learns to distinguish a set of normally occurring patterns (self) from its complement (nonself) when only positive instances of the class are available. In the machine learning literature, this has been studied as the problem of learning from examples of a single class or as concept learning from positive examples [5], [6]. Although in principle this learning problem is solvable for noise-free classes and certain formal domains, in many practical cases, the problem is known to be computationally intractable [7] or to lead to substantial overgeneralization [8]. The statistical community has examined the closely related problems of outlier detection and robust statistics, finding that the effectiveness of the learning system depends critically on domain-dependent assumptions about the distributions of self and nonself data. For example, in a computer security context Lane employed time-series models of user behaviors and a prior bias against false alarms to differentiate intruders from authorized system users [9].

There are many well known change-detection and checksum algorithms which solve a restricted form of the anomaly-detection problem, known as change detection. Here, self is known exactly, is small enough to be stored in a single location, remains constant over time, and can be unambiguously distinguished from nonself. However, for cases in which these assumptions don't hold, the discrimination task is more challenging, and in these situations the negative-detection approach may be appropriate.

Since its introduction in [2], interest in negative detection has been growing, especially for applications in which noticing anomalous patterns is important, including computer virus detection [10], [11], intrusion detection [12]–[14], and industrial milling operations [15]. Recently, other categories of applications have been proposed, including color image classification [16] and collaborative filtering and evolutionary design [17]. Underlying all this work, however, is the question of negative versus positive detection. That is, when is it advantageous to use a negative-detection scheme instead of the more straightforward positive-detection scheme?

To date, there has been little theoretical analysis of the relative merits of positive versus negative detection, and that is the primary goal of this paper. Such analysis is complicated by several factors, which we also address. Most important is the choice of representation and match rule. How is the set of normal patterns represented? How are anomaly detectors represented? What are the criteria for determining when a detector has detected a pattern? The analysis is further complicated if the self set is too large or too distributed to observe completely, if it is nonstationary (self sets that change over time), or if the detectors are changing over time.

In this paper, a formal framework is given for analyzing the tradeoffs between positive and negative detection schemes in terms of the number of detectors needed for maximum coverage. Earlier analyzes focused on the cost of generating detectors (see Section II), rather than on the coverage provided by a complete set of detectors. We then discuss the need for partial matching rules and generalization, as well as what this implies for comparing positive and negative detection. We review the r-contiguous bits (rcb) matching rule, which is the most common match rule used with negative detection. We introduce a related match rule, called *r-chunks*, discuss its advantages, and characterize the generalizations that are induced by it in terms of a concept known as the *crossover closure*, giving the expected size of the crossover closure for r-chunks. We are then in a position to compare quantitatively the positive and negative detection schemes under r-chunks. Finally, we consider a method for increasing the diversity of coverage, known as permutation masks [12], [18], show that permutation masks reduce the number of nondetectable strings in the nonself set, for any fixed detection threshold, and give quantitative results on how many permutations are required to maximize this effect.

One consequence of using an approximate scheme for representing a class of explicit instances (e.g., the self set) is that the class will implicitly contain some instances that were not in the original sample. If the approximation is a good one, then this is an advantage, and the approximation is said to *generalize* from the set of observed patterns to a useful class. In the case of anomaly detection, new observations that are similar to the original sample (self) are classified as part of self, and false positives are reduced. If the generalization is a poor one, new observations will frequently be misclassified, leading to high false-positives and/or false-negatives. In this paper, partial matching rules specify the approximation, and the r parameter controls how large the approximation is. We are interested in characterizing as precisely as possible the kinds of generalizations that are induced by our partial match rules. When we discuss the entire set covered by the approximation we refer to it as the *generalization*. When we discuss individual strings in the generalization that were not in the original sample, we refer to them as *holes* [3]—strings not in S for which valid fixed-r detectors cannot be generated.

II. Related Work

Outlier detection is a problem arising in many machine learning and data mining contexts. Abstractly, two families of approaches can be identified. Outlier detection with respect to probability distributions attempts to identify those events that are in the low density regions of the probability distribution governing the generation of normal events [19]–[22]. Under this approach, which we shall refer to as statistical anomaly detection, the degree of suspicion attached to an event is inversely proportional to the frequency with which the event has been observed historically.

A second family of approaches to outlier detection attempts to define a measure of distance on the event space [23]–[28]. Common distance measures include the classic L-norm and Hamming distances, Manhattan distance (also known as rectilinear, which generalizes Hamming distance to non-binary coordinate systems), and the vector cosine measure of Salton [29]. Under this approach, the degree of suspicion attached to an event is directly proportional to the distance of the event from, for example, the nearest observed normal event or the center of mass of clusters of normal events.

The main difficulty in applying statistical anomaly detection is that the probability distribution is not known exactly and must be estimated from a sample. Often, this sample is sparse, making density estimation highly error prone. In many applications, the majority of events we must judge do not appear in the sample at all, and hence must be classified as "most suspect". The main difficulty in applying distance criteria is in the construction of a measure that reflects a useful metric of similarity. A poor choice of measures results in meaningless classifications.

Often, either implicitly or explicitly, the approaches are combined [30], [31]. For example, data reduction transformations, such as feature selection and value aggregation, collapse distinct events into clusters of events that are to be treated as if they were indistinguishable [32], [33]. An event is assigned the composite density of the cluster into which it falls. In this way, many events are clustered together, all of which are classified as either suspect or not based on the cluster's composite density. The most suspect events will cluster with no previously encountered events (forming a low density cluster of its own).

Statistical anomaly detection is often used on the problem of intrusion detection in computer security, for example [14], [34]–[41]. Intrusion-detection systems vary widely, but they all seek to protect an information system (e.g., a single computer, a database, a server, or a network of computers) from violations of the system's security policy. In the case of anomaly intrusion-detection systems, protection is provided by building a model of the normal (legal) operation of the system and treating discrepancies from the model as anomalies. The model of normal behavior can be based on any observable behavior of the system. Audit logs, patterns of network traffic, user commands, and system calls are all common choices. Such an approach can work well if the anomalies in normal operation are well-correlated with security violations. The extent to which this is true is a topic of current debate in the intrusion-detection community.

In ref. [2], 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* (rcb) match rule [42], [43], 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 computer virus detection, and it was noted that observed performance on real data sets was often significantly better than that predicted by the expected-case analysis. This improved performance was ascribed to regularities in the protected data (self).

Refs [1], [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 most plausible matching rules were also identified. These limits arise because partial matching rules obscure the boundary between the self and nonself sets. Ref. [3] 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 [44]–[46], and included an analysis of the holes in coverage associated with his algorithm. In this work, it was assumed that both the protected data and the detectors are unchanging through time.

Likewise, both projects focused on the problem of generating detectors efficiently, rather than the problem of how many detectors are needed for maximal performance. The latter question is the one addressed in this paper.

Lamont and Harmer applied negative selection to the problem of computer virus detection, focusing on different match rules [11]. They developed a prototype implementation to demonstrate that the approach is both scalable and effective. As part of their work, they evaluated 12 different match rules including Hamming distance and rcb, concluding that the Rogers and Tanimoto rule (an extension of Hamming distance) was the best fit for their application, because it did the best job of balancing the generality and specificity of each individual detector.

Hofmeyr [12], [18], [47] 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 nonstationary) 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 intrusion-detection system featured rolling sets of detectors in the sense that 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 [13]. Their implementation used a significantly more complicated representation than [18], with a much larger alphabet, shorter strings, and more general matching thresholds (lower r in r-contiguous bits matching). Like Hofmeyr, they used the original random-generation method of producing detectors. They reported difficulties generating valid detectors and concluded that negative selection is infeasible on its own for realistic network intrusion detection. Ref. [48] proposes several explanations for their results, based largely on an analysis of the parameter settings that were selected. Williams et al. [14] reports 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 [13], [18], however, in that it monitors UDP and ICMP packets as well as TCP packets.

Dasgupta and Gonzalez used negative detection on the network intrusion-detection problem [49]. They worked with the Lincoln Labs data set [50], used a real-valued representation, and divided the Lincoln Labs data into overlapping intervals. The paper is interesting because it compares the performance of positive and negative characterization of self, reporting that the positive approach is more accurate, but also computationally more expensive. It isn't obvious how meaningful the comparison is, however, because the positive and negative methods are quite different. The positive detection method simply memorizes a set of training (attack-free) points and classifies any test point as "normal" if it is within a given distance of any memorized self-point. The negative-detection method uses a genetic algorithm to evolve complex rules for detecting nonself. Thus, the study could more properly be viewed as comparing evolved rules with a simple memory-based approach.

In addition to intrusion detection, several authors have been interested in using negative detection for other applications. Sathyanath and Sahin used negative detection for color image classification of finished wooden components, specifically kitchen cabinets [16]. Bradley and Tyrrell describe a hardware implementation using field programmable gate arrays (FPGA) [51]–[53]. Their detectors monitor transitions, together with some input/output information, in finite-state machines to detect faults. The implementation uses negative detection and r-contiguous bits match rules. Detectors are generated off-line and the entire self set is known at the time detectors are being generated. Chao and Forrest [17] describe a negative-detection approach to interactive search algorithms, in which subjective evaluations drive the exploration of large parameter spaces. Their algorithm learns what parts of the search space are *not* useful, based on the negative-detection strategy of the natural immune system. The algorithm is capable of finding consensus solutions for parties with different selection criteria.

Sometimes, the set of anomalous patterns is known, or thought to be known, exactly, and a set of signatures is used to cover the dangerous parts of nonself precisely. Examples of this approach are signature-based scanners for intrusion detection [54] and commercial virus-scanners.

III. A Statistical Modeling Framework

In this section we cast the anomaly intrusion-detection problem in a statistical framework. Although the unrestricted version of the framework is quite complex, it is nevertheless useful to begin with the overall view. This helps us make rigorous certain notions which are required later in the paper, and it allows future work to evolve toward meeting broader objectives.

Our basic unit of analysis is a length l binary string, called a packet. At each time step t, a packet $x(t)$ is generated. We assume that $x(t)$ is generated either by the legitimate process G or the malicious process B. In reality, there could be any number of subprocesses contributing to each of these processes (e.g., many legitimate users with different behaviors), but we proceed as if there were a single G and a single B.

More precisely, a sequence $x(1), x(2), \ldots, x(t)$ of length l binary strings (packets) is generated. A hidden process $\mathcal{H}$ governs the generation of the packet $x(t)$ at time t. Random variable $y(t) = 0$ if process G generates $x(t)$ and $y(t) = 1$ if process B generates $x(t)$. In general, it may be the case that the distribution of random variable $y(t)$ depends on the sequences $x(1), x(2), \ldots, x(t-1)$ and $y(1), y(2), \ldots, y(t-1)$ of random variables, as well as on the identity of the particular time step t. In the remainder of this paper, however, we assume that the distribution of $y(t)$ is independent of t, and also of $y(t')$ and $x(t')$ for $t' < t$. Thus, $Pr\{y(t) = 0\}$ and $Pr\{y(t) =$

1} —also referred to as the prior probabilities of the processes $P(G)$ and $P(B)$ respectively—are unchanging over time.

There are several (typically unknown) next packet distributions of potential interest; estimating these distributions, under various assumptions to be introduced below, is a major focus of this paper. The pair of distributions

$$\begin{aligned} Pr\{x(t)|y(t) = 0,& x(t-1), x(t-2), \ldots, x(1), \\ & y(t-1), y(t-2), \ldots, y(1)\} \\ Pr\{x(t)|y(t) = 1,& x(t-1), x(t-2), \ldots, x(1), \\ & y(t-1), y(t-2), \ldots, y(1)\} \end{aligned}$$

specify next packet distributions under G and B respectively, conditioned on knowledge of the first $t-1$ packets $x(t')$ generated, plus a specification (the values of $y(t')$) of which process generated each of these $x(t')$. Estimating these distributions from the data amounts to a supervised learning problem, since the $y(t')$ label the training data $x(1), x(2), \ldots, x(t-1)$, indicating which process generated each packet. Note that these probabilities are related via Bayes Theorem to the posterior distribution on which process G or B generated the current packet $x(t)$. See Section 3.3 for details. Note also that the mixture distribution for the next packet generated is

$$\begin{aligned} Pr\{y(t) = 0\} \cdot Pr\{& x(t)|y(t) = 0, x(t-1), x(t-2), \\ & \ldots, x(1), y(t-1), y(t-2), \ldots, y(1)\} \\ +Pr\{y(t) = 1\} \cdot Pr\{& x(t)|y(t) = 1, x(t-1), x(t-2), \\ & \ldots, x(1), y(t-1), y(t-2), \ldots, y(1)\} \end{aligned}$$

In the applications studied in this paper, we do not have direct knowledge of the values of the random variables $y(t')$—that is, we do not know with certainty which of the processes G or B generated the previous packets in the sequence, leaving us with and unsupervised learning problem. In particular, we will be concerned with estimating the pair of distributions

$$\begin{aligned} &Pr\{x(t)|y(t) = 0, \mathcal{H}, x(t-1), x(t-2), \ldots, x(1)\} \\ &Pr\{x(t)|y(t) = 1, \mathcal{H}, x(t-1), x(t-2), \ldots, x(1)\} \end{aligned}$$

when we have knowledge of the process priors $P(G)$ and $P(B)$ (that is, knowledge of the distribution of the hidden process $\mathcal{H}$), or with estimating the pair of distributions

$$\begin{aligned} &Pr\{x(t)|y(t) = 0, x(t-1), x(t-2), \ldots, x(1)\} \\ &Pr\{x(t)|y(t) = 1, x(t-1), x(t-2), \ldots, x(1)\} \end{aligned}$$

when we do not have knowledge of the process priors.

For some problems, the ultimate objective is to identify short temporal groupings of anomalous packets, or the time intervals most likely to contain packets generated by B. Many intrusion-detection problems have this quality, and the above formulation supports such an analysis. However, there are important special cases in which the analysis shifts from sequences of packets to single packets, and these are the focus here.

The next packet distribution of each process may depend on the current time step t, as well as on the pattern of packets generated at time steps prior to t. This allows the possibility that each of G and B is nonstationary, its distribution depending both on the actual time step t and the identity of packets previously generated by both processes. This nonstationarity might reflect, for example, that

- Composite user behavior changes over time (different populations of users);
- A user's behavior at time t depends on behaviors at times $t' < t$ (a single user's action depends on his or her prior actions);
- The nature of attacks changes over time;
- Changes in the environment (e.g., a new computer added to a network).

We abstract away the question of whether and how the true distributions of G and B change over time by specifying our posterior of the next packet $x(t)$ generated at any instant t in time, given some dynamic $sample(t)$ of G at instant t. That is, in this paper we focus on situations in which it suffices to replace

$$\begin{aligned} & Pr\{x(t)|y(t) = 0, \mathcal{H}, x(t-1), x(t-2), \ldots, x(1)\} \\ \text{or} \quad & Pr\{x(t)|y(t) = 0, x(t-1), x(t-2), \ldots, x(1)\} \\ \text{by} \quad & Pr\{x(t)|y(t) = 0, sample(t), t\}, \end{aligned}$$

where $sample(t)$ is an (unordered) set of packets. $sample(t)$ may include a subset of the packets $x(t'), t' < t$, restricted to packets deemed to have been generated by G (i.e., packets $x(t')$ for which it is deemed likely that $y(t') = 0$), and perhaps further restricted by other criteria (e.g., recency, random sampling). $sample(t)$ may also be seeded with an initial population of packets.

$sample(t)$ reflects our current state of knowledge of G at time t, and the next packet generated by G is dependent only on this knowledge—given $sample(t)$, $x(t)$'s distribution is conditionally independent of all other information (e.g., of other previously generated packets). As $sample(t)$ changes with time (see below), the next packet distribution changes. Whether or not process G actually is changing over time (or whether, for example, our state of knowledge simply is changing) is immaterial to the model. Further, because $sample(t)$ is unordered and assumed to be generated by, or otherwise representative of, G, the detection question of interest is to decide whether the next packet $x(t)$ is generated by G or B—our model does not address the problem of deciding whether a subsequence embedded in $x(1), x(2), \ldots, x(k)$, is generated by G or B.

In the remainder of this paper we consider a simple family of distributions $Pr\{x(t)|y(t) = 0, sample(t), t\}$ as our modeling of how G generates its next packet. Intuitively, we assume a distance measure between an arbitrary packet p_k and the current $sample(t)$. The closer a packet p_k is to $sample(t)$, the greater the probability $Pr\{p_k|y(t) = 0, sample(t), t\}$. Many notions of distance are possible. For example, the distance of a packet p_k from $sample(t)$ could be p_k's minimum Hamming distance to a member of $sample(t)$. Note that this measure does not distinguish between members of $sample(t)$ (assigning each such string a distance of 0), but does differentiate between packets not in $sample(t)$. Alternatively, distance could be based on frequency counts—the "distance" of p_k from $sample(t)$ could be the reciprocal of the number of instances of p_k in $sample(t)$, with $1/0$ defined to be some large constant C. This distance measure fails to distinguish between packets not in $sample(t)$ (assigning each such string a distance of C), but does differentiate among packets in $sample(t)$. Of course, one could develop a hybrid of these two measures in which a distance based on Hamming distance is used for strings outside of $sample(t)$, and a distance based on frequency counts is used for strings in $sample(t)$.

We adopt a simple categorical division into "similar to $sample(t)$" versus "dissimilar from $sample(t)$" and distinguish these distance categories by means of a *generation rule* which attempts to characterize the underlying set from which $sample(t)$ is likely drawn. Unlike the two distance measures mentioned earlier, a generation rule allows only the distinction between likely and unlikely strings under G, which simplifies the detection task considerably. Experimental studies have shown that these simplifications often capture sufficient detail of process behavior to provide effective detection [12], [55].

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

This is the generalization discussed in Section I.

Given a generation rule Q, we wish to specify a distribution to reflect that each member of $Q(sample(t))$ is more likely to be generated at time t by G than is each nonmember. Several simple alternatives exist, though care must be taken to ensure a proper probability distribution results. For example,some applications, it may be appropriate to have the probability of the set $Q(sample(t))$ be some large multiplicative factor f of the probability of the complement of this set. We can construct a probability distribution capturing this relative frequency by first setting $p'_1 = f/|Q(sample(t)|$, $p'_2 = 1/|U - Q(sample(t))|$ and then (noting $p'_1 \cdot |Q(sample(t)| + p'_2 \cdot |U - Q(sample(t))| = f + 1$) normalizing to obtain proper point probabilities for individual strings, that is,

$$Pr\{x(t)|y(t) = 0, sample(t), t\} = \begin{cases} p'_1/(f+1), & if \quad x(t) \in Q(sample(t)) \\ p'_2/(f+1), & if \quad x(t) \notin Q(sample(t)) \end{cases}$$

Assuming there are more strings in the complement of $Q(sample(t))$ than in the set itself, the point probability of each member string would be greater than that of each nonmember string. Alternatively, in other contexts (e.g., when $Q(sample(t))$ is larger than its complement), it would be necessary to specify directly that the common point probability of each member of $Q(sample(t))$ is some large multiple of the common point probability of each nonmember, and then again normalize to proper probabilities by taking into account the number of members and nonmembers.

Whatever method is used to set the distribution, we assume $Pr\{x(t)|y(t) = 0, sample(t), t\}$ is specified by

$$Pr\{x(t)|y(t) = 0, sample(t), t\} = \begin{cases} p_1, & if \quad x(t) \in Q(sample(t)) \\ p_2, & if \quad x(t) \notin Q(sample(t)) \end{cases}$$

where p_1 and p_2 satisfy $0 \le p_2 < p_1 \le 1.0$, and $p_1 + p_2 = 1.0$.

As a final comment, we note that, under this modeling, although the generation rule Q and measure of distance (e.g., member or not of the set $Q(S)$) are assumed fixed and known exactly (as opposed to being dependent on the data), the probabilities assigned to $x(t)$ under G do change as the sample changes with time. In particular, since $sample(t)$ typically is dependent on previously observed packets—evolving with our knowledge of the true image of process G—our distribution of the next packet generated changes as the data changes.

A. Example Generating Rules

We now present three sample generation rules which can be used to characterize the underlying set from which $sample(t)$ is drawn. First, is Hamming radius, which generalizes $sample(t)$ to strings within a given Hamming distance of strings in the sample. Next, we consider crossover closure, which generalizes $sample(t)$ to strings which can be constructed by pasting together the windows of strings appearing in $sample(t)$. Finally, we examine n-gram matching, which generalizes $sample(t)$ to strings whose n-grams have been observed in the $sample(t)$.

1) Example: Hamming Radius: The Hamming distance measure discussed above can be simplified to a categorical measure based on a threshold T. In particular, the generation rule $Hd_T(S)$ is defined to be the set of strings within a Hamming distance of T or less to some member of S. The use of this generation rule in detection is considered in Section VI. Hamming distance match rules have been used with negative detection in the context of immunological modeling, e.g., [56].

2) Example: Crossover Closure: The generation rule which we consider in the most detail is called "crossover closure." As we will show in subsequent sections, the crossover closure is closely related to the two match rules most commonly used in conjunction with negative detection.

Given a set S of strings, and a fixed $1 \le r \le l$, the crossover closure $CC_r(S)$ of S is defined in terms of its length r windows (i.e., r consecutive string positions) as:

$$CC_r(S) = \{u \in U | (\forall windows\ w)(\exists s \in S)\ u[w] = s[w]\}$$

In words, string $u \in U$ is in the crossover closure of S if and only if each of u's t windows exactly matches the corresponding window of some member of S. When S is such that $CC_r(S) = S$ we say that S is *closed* under crossover closure. As we shall see, the class of sets closed under crossover closure have a central role in our methods.

One interesting motivation for crossover closure comes from relational database theory [57], [58]. The strings of a sample $sample(t)$ can be viewed as the tuples of the current instance of a relation scheme $R(A_1, \ldots, A_l)$, where each attribute A_i corresponds to packet position i and has domain $\{0, 1\}$. For example, the $sample(t) = \{0000, 1011\}$ can be represented as a relation scheme $R(A_1, A_2, A_3, A_4)$ whose current instance is shown below:

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

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

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

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

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

In order to reconstruct the original instance of R from these projections onto the R_i, one computes the natural join of the R_i. However, it is not always the case that the join of the projection recovers the original instance of R—in fact, the join of the projected instances is precisely the crossover closure of the set of tuples (strings) in the original instance of R. In our example, the join $R_1|X|R_2|X|R_3$ of the instances shown above results in the following instance of R, which can be seen to be the crossover closure of the strings in the original instance of R.

$R_1|X|R_2|X|R_3 =$

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

Relational database theory has exactly characterized when the projections of R join to recover the original instance (see Section VII-B and [57], [58]). This is known as the lossless join condition. The lossless join condition thus also characterizes exactly when a set of packets is equal to its crossover closure. One interpretation of this correspondence is that, since many naturally occurring collections of information do in fact satisfy the lossless join condition, it is plausible that in many contexts the most likely set of strings to be generated by G is closed under crossover closure. When this is the case, $sample(t)$—if it is a representative sample of strings generated by G—can be interpreted as being a sample drawn from $CC(sample(t))$, or from a larger set containing $CC(sample(t))$ and itself closed under crossover. In such situations, it is appropriate to deem packets which are members of $CC(sample(t))$ as relatively likely under G.

3) Example: n-grams: The n-gram matching rule has been used in a wide variety of settings including natural language processing [59], document classification [60], and program monitoring [55], [61]–[63] for intrusion detection. In the latter application, n-grams can be applied when the behavior of the protected process (for example, an executing computer program) can be represented as a sequence of letters taken from some alphabet. A sample of normal behavior S is a collection of such sequences, called a "trace" collected under normal operating conditions. Different traces can be of different lengths and are parsed using a sliding window of size n. Each substring of length n (the n-gram) is stored in the program's profile.[2] Once the profile is complete, a new execution is analyzed in terms of the n-grams composing its trace. Any n-gram that occurs during execution, which does not appear in the profile, constitutes an anomaly.

In order to characterize the generation rule for n-grams we first restrict our attention to the generalization induced over traces of length l and create a directed acyclic graph (DAG) with $l - n + 1$ levels, where each level has a node for each distinct observed n-gram. A node with label P in level i is connected to a node with label Q in level $i+1$ if the last $n-1$ symbols of P match the first $n-1$ symbols of Q (akin to the previous example). A similar graphical construction was used in [66] for determining multiple length n-grams. The set of protected traces of length l can be retrieved by traversing all possible paths, form the first to the last level, in this graph. We write the generalization induced for traces of size l under n-grams as $CC_n^l(S)$: The crossover closure at l (see example 2). Finally, restricting traces to specific lengths, the generalization achieved by n-gram matching can be expressed as:

$$CC_n^*(S) = \bigcup_{l \in L} CC_n^l(S)$$

where L is the set of acceptable lengths.

There are two differences between this generation rule and that presented in Section III-A.2. In the latter we considered all strings to be of the same specific length l, a small assumption that could easily be relaxed. More importantly, in Section III-A.2 we accept only those patterns that have been previously observed at a particular window position, whereas with n-grams we disregard the information about window location.

B. Computing Posterior Probabilities on Intrusion Events

In general, deciding whether $x(t)$ is generated by G or B requires (1) distributions for B as well as G, and (2) process priors $P(G)$ and $P(B)$. In previous work [32], [38], [67], we considered models in which $Pr\{x(t)|y(t) = 1, sample(t), t\}$ is uniform over all strings in U versus models for specific types of malicious behavior. Modeling $Pr\{x(t)|y(t) = 1, sample(t), t\}$ as uniform over all U is a conservative approach, consistent with the view that we have little prior information about likely forms of attack. Modeling B as uniform places a constant c in the numerator of the likelihood ratio

$$\frac{Pr\{x(t)|y(t) = 1, sample(t), t\}}{Pr\{x(t)|y(t) = 0, sample(t), t\}}$$

and hence equates the posterior probability

$$\begin{aligned} & Pr\{y(t) = 1|x(t), sample(t), t\} \\ = & \frac{Pr\{x(t)|y(t) = 1, sample(t), t\} \cdot P(B)}{Pr\{x(t)|sample(t), t\}} \\ = & \frac{c \cdot P(B)}{c \cdot P(B) + Pr\{x(t)|y(t) = 0, sample(t), t\} \cdot P(G)} \end{aligned}$$

with the degree of $x(t)$'s rareness under G. (With the numerator of the expression for the posterior $Pr\{y(t) = 1|x(t), sample(t), t\}$ fixed at constant $c \cdot P(B)$, only $Pr\{x(t)|y(t) = 0, sample(t), t\}$ varies as $x(t)$ varies.) We identified this approach with what the literature commonly refers to as *anomaly detection*, and reported experimental results with examples both when anomaly detection is sufficient and when it is not (and specific models of malicious behavior are necessary for satisfactory detector performance).

Here, we assume that B is uniform. With this assumption, we can either specify process priors $P(G)$ and $P(B)$ and a detection threshold on the posterior $Pr\{y(t) = 1|x(t), sample(t), t\}$, or, as in [32], [67], we can simply prioritize the packets by the likelihood they were generated by B, that is, rank the packets by the values of their likelihood ratios—this gives an ordering equivalent to ranking by the posterior $Pr\{y(t) = 1|x(t), sample(t), t\}$, regardless of the values of the priors $P(G)$ and $P(B)$. With constant c in the numerator of the likelihood ratio, and the G process modeled as described in the previous section, ranking packets by their likelihood ratios reduces to partitioning packets into two classes: those that are most likely generated by G and those most likely not generated by G. We are interested in compact schemes which allow us to distinguish quickly between low and high probability anomalous packets. Under the family of distributions proposed above, this task is equivalent to distinguishing, for a given generation rule Q and current sample $sample(t)$, between packets contained in $Q(sample(t))$ and those outside this set. In the following sections we present positive and negative detection schemes for addressing this problem. We also compare the time and space efficiencies of several variants of these schemes. All of these schemes use simple and efficient string matching rules for deciding whether or not a packet p_k is a member of $Q(sample(t))$.

IV. A Taxonomy of Detection Schemes

This section explores several detection schemes based on the framework outlined above. Given a generation rule Q and a current sample, $sample(t)$, an instance of a detection scheme (known simply as a detection algorithm) must be able to decide whether or not a packet p_k is a member of $Q(sample(t))$. That is, a detection algorithm can be viewed as protecting the subset $Q(sample(t)) \subset U$. Strings within $Q(sample(t))$ are regarded as likely to be part of self, while those in $U - Q(sample(t))$ are most likely to be anomalous. From the perspective of formal languages, a detection scheme is a language recognition model. And, analogous to the issues arising in the study of formal languages, we are interested in how to construct a detection scheme that can protect the class of languages implied by a particular generation rule of interest, and also in determining the class of languages that various detection schemes can protect.

Many detection schemes can be constructed from the simple building blocks outlined so far—the simple string matching rules and the alternative interpretations associated with a match. Before proceeding to analyze specific cases of interest, e.g., [12], we first outline a systematic taxonomy for all the schemes. This taxonomy consists of three dimensions:

- The form of a single detector together with its match rule—a binary relation on detectors and packets, specifying when a single detector and single packet match.
- Whether positive or negative detection is employed.
- Disjunctive versus conjunctive matchings: Whether a single match is sufficient to trigger a decision (positive or negative), or whether multiple matches are required.

In the following sections, we study the relative power of the detection schemes that can be constructed from different choices along the above dimensions. We first compare the classes of languages that can be protected by the schemes, and then consider the resources (time and space) required by the schemes to protect their languages.

A. Form of Detector and Matching Rule

A single detector, as considered in this paper, is either:

1) An element of U, or
2) An r-chunk, a length r binary string along with a specified window position.

When a detector d is an element of U, we are interested primarily in the rcb match rule: that is, for $d, x \in U$

$$dMx \leftrightarrow (\exists\ window\ w)(d[w] = x[w]).$$

We refer to such detectors under the rcb match rule as rcb detectors. (In Section VI, we contrast the rcb match rule with Hamming distance, an alternative match rule for detectors in U.) If d is an r-chunk on window w, the matching rule we consider is: for $x \in U$

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

B. Detector Operation

Let Υ be a set of rcb or r-chunk detectors and assume the corresponding matching rule from the previous subsection. We now consider the interpretation given to a match between a packet and a detector. In particular, does a match mean the packet is "in the language" (positive detection, denoted by P) or "outside the language" (negative detection, denoted by N)? Further, do we require a single match anywhere in the packet (disjunctive matching, denoted by D), or a match in all packet window positions (conjunctive matching, denoted by C)? The four combinations of possible answers specify four corresponding detection schemes: PC, NC, PD, ND.

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

The language (subset of U) "accepted" by each of the four detection schemes when instantiated with a fixed set Υ of rcb or r-chunk detectors is defined as follows:

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

4) Negative Conjunctive Detection $Scheme_{NC}$: $Scheme_{NC}(\Upsilon)$ is the set of strings x in U such that $(\exists window\ w)(\nexists d \in \Upsilon)(dMx)$.

C. r-chunks Matching Subsumes rcb Matching

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

Lemma 3.1: If d is an rcb detector, there exists a collection $T(d)$ of r-chunk detectors such that $\forall x \in U, d[w] = x[w] \leftrightarrow \exists\ t \in T(d),\ t = x[w]$.

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

To illustrate the construction of $T(d)$ used in the proof of the previous lemma, let $d = 1011$ be an r-contiguous bit detector with l=4, r=2. Then $T(d) = \{d[1], d[2], d[3]\}$ as shown below is the equivalent set of r-chunk detectors:

d:	1	0	1	1
$d[1]$:	1	0		
$d[2]$:		0	1	
$d[3]$:			1	1

It follows from the lemma that, for each of the four detection schemes defined above, any set $L \subset U$ of strings that can be recognized with a set of rcb detectors can be recognized with some set of r-chunk detectors. In particular, we have:

Theorem 3.1: If Υ is a set of rcb detectors, and α and β are any choices for detector dimensions as above, then $Scheme_{\alpha,\beta}(\Upsilon)$ = $Scheme_{\alpha,\beta}(T(\Upsilon))$.

Proof: It follows from lemma 3.1 that for any of the membership predicates $F_{\alpha,\beta}$ associated with any of the four detection schemes $Scheme_{\alpha,\beta}$, and for every string $x \in U$, [Υ satisfies $F_{\alpha,\beta}$ $wrt\ x$] $\leftrightarrow$ [$T(\Upsilon)$ satisfies $F_{\alpha,\beta}$ $wrt\ x$]. ■

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

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

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

D. Detection Schemes and the Crossover Closure

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

Theorem 4.1: Let Υ be any set of rcb or r-chunks detectors and let Υ' denote the complement of Υ relative to the universe of all rcb or r-chunks detectors over the same alphabet and of the same length as the detectors in Υ. Similarly, the complement of the set $Scheme_{\alpha,\beta}(\Upsilon)$ (which is a subset of U) is taken relative to U.

Then

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

Further, the subset containment is proper for some Υ.

It follows from $Scheme_{ND}(\Upsilon) = Scheme_{PC}(\Upsilon')$ that the class of languages recognized by $Scheme_{ND}$ is identical to the class recognized by $Scheme_{PC}$, when either rcb or r-chunk detectors are considered. Similarly, it follows from $Scheme_{PD}(\Upsilon') = Scheme_{NC}(\Upsilon)$ that the class of languages recognized by $Scheme_{PD}$ is identical to the class recognized by $Scheme_{NC}(\Upsilon)$, when either rcb or r-chunk detectors are considered.

As discussed in Section III, one of our most important detection applications is to protect the crossover closure of a sample $sample(t)$. There is a strong relationship between crossover closure and the schemes $Scheme_{ND}$ and $Scheme_{PC}$ when r-chunk detectors are used.

Let $sample(t)$ be any subset of U. Let W_P denote the set of windows present in $sample(t)$, that is, the union of the projections of $sample(t)$ onto each window (the window size is fixed and understood). Let W_N denote the set of windows not present in $sample(t)$, that is, $W_N = W_P'$. We then have:

Theorem 4.2: $Scheme_{PC}(W_P) = Scheme_{ND}(W_P') = Scheme_{ND}(W_N) = CC(sample(t))$.

Proof: $x \in Scheme_{PC}(W_p) \leftrightarrow \forall_w\ x[w] \in W_p \leftrightarrow \forall_w \exists\ s \in sample(t), x[w] = s[w] \leftrightarrow x \in CC(sample(t))$ ■

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

Theorem 4.3: The class of languages recognized by $Scheme_{PC}$ and $Scheme_{ND}$ when r-chunk detectors are used is exactly the class of sets closed under crossover closure.

Proof: If a set A is closed under crossover closure i.e. $A = CC(A)$ then we can construct a set of detectors W_P from the windows present in A. It follows from theorem 4.2 that $Scheme_{PD}(W_P) = Scheme_{ND}(W_P') = CC(A)$. By the definition of $Scheme_{PC}$ $x \in Scheme_{PC}(\Upsilon)$ if for all windows w $x[w] \in \Upsilon$, where Υ is an arbitrary set of detectors. We construct a set of detectors W_P by taking, for every window w and every $x \in Scheme_{PC}(\Upsilon)$, the projections $x[w]$ such that $x[w] \in \Upsilon \leftrightarrow x[w] \in W_P$. It follows that $x \in Scheme_{PC}(\Upsilon) \leftrightarrow x \in Scheme_{PC}(W_p)$. Finally, using theorem 4.2, we have $Scheme_{PC}(\Upsilon) = Scheme_{PC}(W_P) = Scheme_{ND}(W'_P) = CC(Scheme_{PC}(\Upsilon))$. ∎

Results of the previous section imply that, when rcb detectors are used, $Scheme_{PC}$ and $Scheme_{ND}$ can recognize only sets closed under crossover closure, but not all sets closed under crossover closure, since the class of languages recognized by these schemes when rcb detectors are used is properly contained by the class recognized when r-chunks are used.

Finally, consider the schemes $Scheme_{PD}$ and $Scheme_{NC}$ using r-chunks. These schemes do not recognize all sets closed under crossover closure and also do recognize some sets not closed under crossover closure. To see this, we first observe that there exist sets S such that S is closed under crossover but S' is not. For example, let l=3, r=2, $S = \{000\}$. $CC(S) = S$, and hence S is closed under crossover. S' contains the other seven length 3 strings and $CC(S') = U$.

Therefore:

1) $Scheme_{PD}$ and $Scheme_{NC}$ cannot recognize all sets closed under crossover, because if $Scheme_{PD}$ recognizes a set S closed under crossover such that S' is not closed under crossover, $Scheme_{ND}(\Upsilon) = (Scheme_{PD}(\Upsilon))'$ implies $Scheme_{ND}$ recognizes S'.
2) $Scheme_{PD}$ and $Scheme_{NC}$ can recognize some sets not closed under crossover because $Scheme_{ND}$ can recognize a set S closed under crossover such that S' is not closed under crossover, and $Scheme_{ND}(\Upsilon) = (Scheme_{PD}(\Upsilon))'$.

V. Partial Matching and Generalization

This section is devoted to $Scheme_{PC}$ and $Scheme_{ND}$ under the r-chunks matching rule. These schemes both recognize the sets closed under crossover, but they have different properties in terms of implementation requirements. These differences include distributivity, scalability, and algorithm efficiencies. This section explores the size of the detector set Υ and determines its expected size as a function of the number of strings in a randomly generated sample S.

A. *Expected Number of Unique Detectors under r-chunks*

Given a set S it is straightforward to compute exactly how many detectors will be generated for maximal protection, both for the positive and negative detection schemes ($Scheme_{PC}$ and $Scheme_{ND}$) using the obvious generation method. For $Scheme_{PC}$, it requires counting the number of distinct patterns for each of the t windows that comprise the strings in S, whereas for $Scheme_{ND}$, enumerating the distinct patterns that are not present in each window will result in the number of detectors. To provide an estimate of the average number of detectors, for both cases, as a function of the size of S, we note the following:

- The number of strings in U 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

  $$1 - \binom{|S|}{0}(2^{-r})^0(1-2^{-r})^{|S|} = 1 - (1-2^{-r})^{|S|}$$

  The equation is approximate because it considers trials to be independent and sampling to take place with replacement.
- Following the previous item, the expected number of distinct patterns E_r, for a given window of size r, is approximated by $E_r \approxeq 2^r - 2^r(1-2^{-r})^{|S|}$.

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

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

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

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

To establish when one scheme requires a smaller set of detectors than the other for maximal coverage, we determine the number of strings in S for which both schemes yield the same number of detectors i.e. when $E_{pos} = E_{neg}$ (see Figure 2):

$$E_{pos} = E_{neg} = t(2^r(1-2^{-r})^{|S|}) = t(2^r - 2^r(1-2^{-r})^{|S|})$$

Solving for $|S|$

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

Although we are primarily concerned with binary strings, it is worthwhile to note that for an arbitrary size alphabet $\mathbb{A}$ the above equation generalizes to $|S| \approxeq (0.693)|\mathbb{A}|^r$, which affects the point at which one scheme is preferable to the other, in terms of the size of the detector set.

In a worst case scenario, $Scheme_{PC}$ may require $t2^r$ detectors when $|S| \geq 2^r$. Similarly, $Scheme_{ND}$ can also yield up to $t2^r$ detectors but only when there are no self strings whatsoever.

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

- Regardless of the composition of S, a detector must be generated for each pattern in the first window that is not present is S.

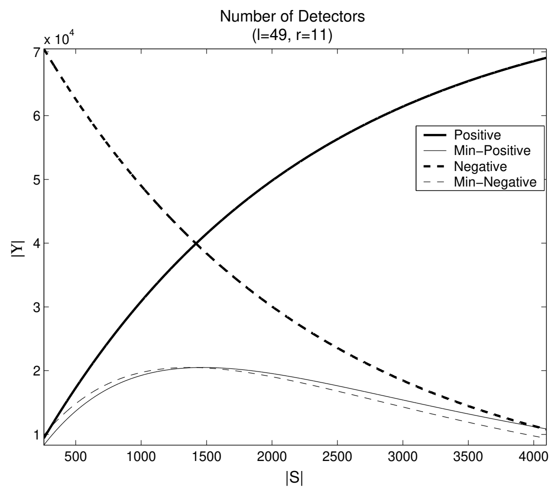


Fig. 2. Number of positive and negative detectors as a function of the size of the self set, assuming the self set was generated randomly. The plot shows both complete and reduced detector sets.

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

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

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

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 required detectors, will exhibit only one of every pair of patterns $w1 = v_i \dots v_{r+i-2}a$, $w2 = v_i \dots v_{r+i-2}\bar{a}$ for each window. This results from a maximum of 2^{r-1} distinct self strings for a detector set size of $t2^{r-1}$. Figure 2 shows the plots of the expected number of detectors for both the full and reduced detector sets.

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

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

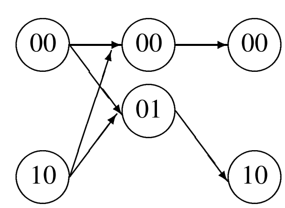
The size of the sample S for which E_{minN} and E_{minP} yield the same number of detectors is the same as with the full repertoire i.e. E_{neg} and E_{pos} eq.(3).

One subtlety about this analysis is that if not all positive detectors are represented explicitly, then some additional information is required to identify implicit matches. This could be stored explicitly, requiring at the very least one bit per implicit match or, it could be derived at runtime by determining, once a match (or implicit match) at window i has been established, if both $w1$ and $w2$ are absent in window $i+1$ [3]. A plot of E_{minP} for different sample sizes is presented in Figure 2 without regards to the extra information required to determine implicit matches.

B. Crossover Closure and its Expected Size

One useful way to visualize the generalization induced by the r-chunks detectors, is to construct DAG $G = (V, E)$ where $|V|$ is the number of distinct bit patterns of length r over all windows t of strings in S. Vertices are labeled by a pair (i, j), where i represents the window number and j stands for the bit pattern. $|E|$ is the number of edges, an edge $e((i, j), (i', j')) \in E \leftrightarrow i' = i + 1$ and the last $r - 1$ bits of pattern j match the first $r - 1$ bits of pattern j' where $(i, j), (i', j') \in V$. Under this representation, the crossover closure is exactly the set of strings formed by all traversals of the graph form level 1 to level t.

Take, for instance, a self set S comprised of the following two strings $S = \{0000, 1010\}$ with $l = 4$, $r = 2$. The corresponding graph $G = (V, E)$ is: $V = \{(1, 00), (1, 10), (2, 00), (2, 01), (3, 00), (3, 10)\}$ 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 visual depiction of the graph (omitting the window number from the vertex labels) is:[4]



Recovering the strings by traversing the preceding graph we find that $CC(S) = \{0000, 0010, 1000, 1010\}$. In order to determine the size of CC for a randomly generated sample, we note that the number of substrings of length l that contain pattern $v_i \dots v_{r+i-1}$ in window w_i is double the number of substrings of length $l - 1$ that contain the pattern in the same window if there are strings in S that exhibit both $v_{i+1} \dots v_{r+i-1}a$ and $v_{i+1} \dots v_{r+i-1}\bar{a}$ in window w_{i+1}, and stays the same if exactly one of these is present. In terms of our DAG representation, the number of paths that include a node with a given label doubles when such a node has two outgoing

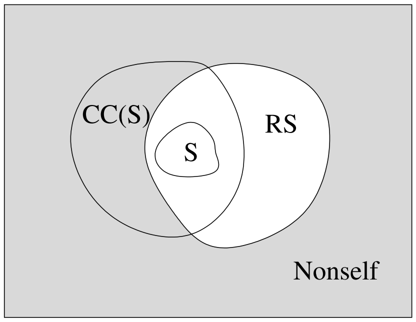


Fig. 3. Crossover closure given a sample S of real self data RS.

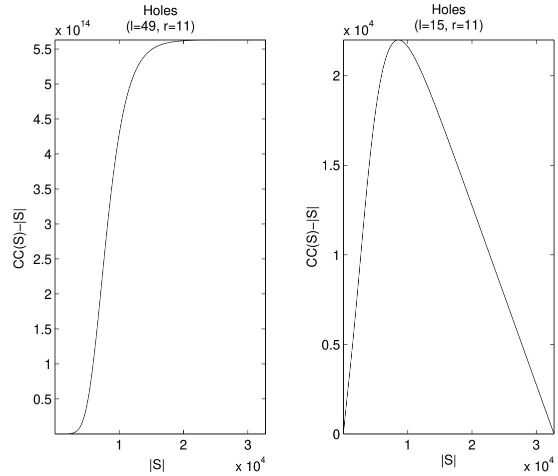


Fig. 4. Number of holes as a function of self-sample size, for two parameter settings. The graph on the left illustrates how fast the generalization increases when r is small compared to the string length. The graph on the right shows how, after the generalization has reached its maximum, the number of holes slowly decreases as more strings are added to the sample S.

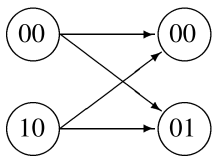
edges. We can write this as a recurrence on the number of windows t:

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

Solving the recurrence yields:

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

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



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

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

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

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

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

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

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

The number of holes $|H|$ can be derived by simply subtracting $|S|$ from $CC(t)$:

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

The size of the generalization, $CC(S)$, can be calculated from the size of S or from the size of the detector set (obtaining an estimate for $|S|$ from either eq. 2 or eq. 1 and substituting in eq. 6). It is important to note that the actual size will depend on the structure of the specific self set. Nevertheless, the analysis provides insight into its behavior (see Figure 4) and enables us to ascertain the impact of allowing novel strings into the sample. This can be useful for determining, in a dynamic scenario, when (or at what rate) should detectors be added or deleted from the working set.

VI. Permutations and Diversity

In refs. [12], [18], an additional mechanism was introduced to improve discrimination between self and nonself. This mechanism is loosely modeled after the diversity of Major Histocompatibility Complex (MHC) molecules used in the natural immune system. In the artificial setting, multiple permutations of the l-length strings (both data and detectors) are stored. This, combined with the contiguous-bits match rules, provides the system with *diversity* of representation and improves the discrimination abilities of the system.

A. Permutation Maps (π) for Reducing the Number of Holes.

If H, the set of holes, is too large, accurate discrimination between RS and $U - RS$ 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.[5] By controlling the number of permutations used, he was able to achieve greater control over the r-contiguous bits generalization. One interesting property of this transformation is that if we consider all non-redundant permutation masks, the r-contiguous bits matching rule is able to protect strictly more sets than Hamming Distance.

Theorem 1.1: The class of languages recognized by r-contiguous bits, with permutation masks, properly contains that recognized by Hamming distance.

Proof: Let:

- x, d be strings of length l, d is a detector.
- $Rcb(x,d) : Strings \times Strings \rightarrow \mathbb{N}$ 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 $Rcb(x,d) \geq r$.
- $Hd(x,d) : Strings \times Strings \rightarrow \mathbb{N}$ 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, $Hd(x,d) \leq l-r$.
- $\pi(x)$ denote some permutation of string x.
- $L(Rcb)$ denote the class of languages recognized by Rcb with permutation masks for a given r and a detector set Υ.
- $L(Hd)$ denote the class of languages recognized by Hd given r and a set of detectors Υ.

The class of languages recognized by Rcb, with permutation masks, contains the class of languages recognized by Hd:

1) $Hd(x,d) > l-r \rightarrow Rcb(\pi(x),\pi(d)) < r$ for all permutations π, and hence: [d a valid detector under Hd] implies [d matches no member of $\pi(L(Rcb))$ under Rcb and any permutation π].
2) Further, if d matches a string x under Hd, we show there exists a permutation π^* such that d matches $\pi^*(x)$ under Rcb, that is $L(Hd) \subset L(Rcb)$. Label the bits that differ between x and d as $c_1c_2 \ldots c_n$, $1 \leq n \leq Hd(x,d)$ and the bits that match as $c_{n+1}c_{n+2} \ldots c_m$, $n+1 \leq m \leq l$. We define π^* as $c_1c_2 \ldots c_nc_{n+1} \ldots c_l$. Clearly π^* is a valid permutation and $Rcb(\pi^*(x),\pi^*(d)) \geq r$. By (1) d is a valid detector for $\pi^*(L(Rcb))$ that matches $\pi^*(x)$. Hence, if L is a language recognized by some set of detectors under Hd, we have shown how to construct an associated set of permutations such that this same set of detectors recognizes exactly L also under Rcb.

∎

In contrast, Hd does not recognize all languages recognized by Rcb with permutation masks, because a detector d under Hd might match more than it matches under a single permutation and Rcb, and thus fail to be a valid detector under Hd.

Example: Let $S = \{011, 110, 100\}$, l=3, r=2, h=000. Under Rcb, we can generate a valid detector d=001 to match h, whereas under Hd the potential detectors for h are $000, 001, 010, 100$ all of which have a Hamming distance less than two from h, but also from all strings in S. Therefore h is undetectable under Hd but detectable under Rcb with permutation masks, hence $L(Rcb) \not\subset L(Hd)$.

For those sets that Hamming distance can recognize, the Rcb rule may require multiple sets of detectors Υ, each with its own transformation π. This mechanism is suitable for a detection scheme that requires precise control over the generalization or that must be distributed.

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

TABLE I

UNDETECTABLE STRINGS UNDER EVERY PERMUTATION, FOR A GIVEN SET S AND THE rcb MATCHING RULE.

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

$s1$= $\boxed{1}\boxed{0}\boxed{1}$ $h1$= $\boxed{1}\boxed{0}\boxed{0}$

$s2$= $\boxed{0}\boxed{0}\boxed{0}$ $h2$= $\boxed{0}\boxed{0}\boxed{1}$

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

$\pi(s1)$= $\boxed{1}\boxed{1}\boxed{0}$ $\pi(h1)$= $\boxed{1}\boxed{0}\boxed{0}$ $d1$= $\boxed{1}\boxed{0}\boxed{1}$

$\pi(s2)$= $\boxed{0}\boxed{0}\boxed{0}$ $\pi(h2)$= $\boxed{0}\boxed{1}\boxed{0}$ $d2$= $\boxed{0}\boxed{1}\boxed{1}$

B. Completeness

We can now ask whether there are languages that cannot be recognized by rcb even if permutation masks are used. That is, are there still strings in nonself (U-RS) which go undetected even if every permutation mask is employed? The answer to this question is yes; as pointed out in ref. [3] all practical match rules with a constant matching probability will exhibit holes.

We show, by construction, that there are languages that cannot be recognized under rcb matching with permutations: Let l=3, r=2 and S=$\{$010,011,100,101$\}$ and consider the hole h=110. Table I 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 I, under each permutation there are no available templates (that is, templates that don't match some string in S), and thus, neither rcb $Scheme_{ND}$ nor $Scheme_{PC}$ can accept the strings in S while excluding h. Although we can construct such examples, it is difficult to characterize the set of all such holes under permutations.

Lifting the restriction that detectors be of length l enables a r-chunk detector to match the string h from the previous example. Nevertheless, there are languages that cannot be recognized under the r-chunks matching rule ($Scheme_{PC}$ and $Scheme_{ND}$) augmented with permutation masks. Consider the following scenario: Let l=3, r=2, S=$\{$111,010,100$\}$ and h=110. Table II lists all 6 permutations of S and h and the

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

TABLE II
UNDETECTABLE STRINGS UNDER EVERY PERMUTATION, FOR A GIVEN SET S AND THE r-CHUNKS MATCHING RULE.

possible r-chunks detectors needed to match h. In general a string h cannot be detected by $Scheme_{PC}$ and $Scheme_{ND}$ under any permutation if all of its template combinations are in self. The template combinations of a string are the templates (strings with unspecified bits) that result from all the possible ways of specifying only r bits. For instance the template combinations of $h = 110$ are $\{11*, 1*0, *10\}$ where $*$ denotes "don't care". It is easy to see that only if one such template does not exist in S, can there exist a permutation that places these bit positions contiguously, without creating the same contiguous bit pattern in S.

VII. DISCUSSION

The preceding sections developed a formal framework for studying tradeoffs between negative and positive detection, and presented several theoretical results that we believe are important to a wide variety of practical problems. In this section, we explore some of the implications and potential extensions of our theoretical results.

A. Computer Security Applications

Over the past several years, a number of anomaly intrusion-detection systems have been developed for computer security, which explore different instances of the detection schemes described in this paper. These systems were reviewed in Section II; some use positive detection [55], [64], [65] and some use negative detection [2], [12]. Likewise, some of these systems use r-contiguous bits matching [2], [12], some use r-chunks [68], some use n-gram matching [55], and some use a variant of n-gram matching known as "lookahead pairs" [64], [65]. Although the negative-selection strategy of the immune system has received a great deal of attention, it should be noted that the immune system also uses positive selection, combining it advantageously with negative selection.

In some cases we have good experimental evidence for preferring one scheme over another. For example, n-grams outperformed lookahead pairs on our early data sets in terms of absolute discrimination ability (unpublished). However, when efficiency matters, as in the case of on-line detection, then the lookahead-pairs method is a clear winner, paying a small penalty in terms of discrimination ability. A second example occurred when we compared rcb detection to r-chunks on a network intrusion-detection task [48], [68]. Such results are anecdotal in the sense that they were obtained by experimentally testing one or two methods on a limited number of data sets. Without some theory to guide us, it is difficult to determine how much a given result depends on the particular data set used and how much it depends on the choice of method. Likewise, it has been difficult to determine which aspects of a given detection scheme are most responsible for its success (or failure) (e.g., negative detection, match rule, or parameter settings).

The results presented in this paper allow us to begin approaching such questions from a theoretical perspective. In particular, the notion of a crossover closure, and its relation to rcb and r-chunks matching, will allow us to understand more deeply when and why these matching methods are preferable to more familiar methods such as Hamming Distance. Likewise, the closed form expressions for detector set sizes, both for positive and negative detection, allow us for the first time to predict how large a problem must be before it pays to use a negative-detection scheme. Until now, the negative-detection approach has been somewhat of a curiosity, notorious as much for its immunological metaphor as for its demonstrated advantages over other methods. Results such as those presented here will form the basis of a more objective evaluation.

B. Relational Databases

Section III introduced the relationship between crossover closure and relational database theory. Specifically, we showed that the crossover closure of a set S of strings is equivalent to the natural join of those strings (interpreted as tuples) projected onto a relational decomposition scheme. There are several aspects of this equivalence which we find interesting.

First, we can characterize when a set of strings is closed under crossover closure in terms of well-studied structural properties known as database dependencies [57], [58]. A set of strings is closed under crossover closure exactly when it decomposes losslessly into the relation schemes

$$R_1(A_1, \ldots, A_r), R_2(A_2, \ldots, A_{r+1}), \ldots,$$
$$R_i(A_i, \ldots, A_{r+i-1}), \ldots, R_t(A_t, \ldots, A_L)$$

induced by its t r-length windows, as we showed in Section III-A. Although the structural characterization of a lossless join decomposition is complex in the general case, the specialized form of the above decomposition simplifies the condition considerably—the characterization reduces to the adherence to a collection of simple multi-valued dependencies on the original relation scheme $R(A_1, A_2, \ldots, A_L)$. In order for the decomposition to be lossless, one of the following multi-valued dependencies on R must hold for each $i = 1, 2, \ldots, (t-1)$:

$A_{i+1}, \ldots, A_{r+i-1} \twoheadrightarrow A_i$, or
$A_{i+1}, \ldots, A_{r+i-1} \twoheadrightarrow A_{r+i}$.

Note that, for example, the multi-valued dependency $A_{i+1}, \ldots, A_{r+i-1} \twoheadrightarrow A_i$ asserts that if R contains the tuples
$a_1 \ldots a_{i-1} a_i a_{i+1} a_{i+2} \ldots a_{r+i-1} a_{r+i} \ldots a_L$ and
$b_1 \ldots b_{i-1} b_i a_{i+1} a_{i+2} \ldots a_{r+i-i} b_{r+i} \ldots b_L$
then it must also contain the tuples
$a_1 \ldots a_{i-1} b_i a_{i+1} a_{i+2} \ldots a_{r+i-1} a_{r+i} \ldots a_L$ and
$b_1 \ldots b_{i-1} a_i a_{i+1} a_{i+2} \ldots a_{r+i-1} b_{r+i} \ldots b_L$

This condition describes the semantics of collections of information that occur naturally within a vast number of diverse application domains. Whether such dependencies arise naturally among sets of strings we wish to protect remains to be investigated.

Taking the connection one step further, we can investigate the correspondence between the schemes $Scheme_{ND}$ and $Scheme_{PC}$, which recognize the class of sets closed under crossover closure, and query systems for relational databases. In particular, if one is able recognize with a set of detectors (possibly significantly pruned to remove redundancy as described in Section V-A.2) a given set closed under crossover closure, one can also represent by the same means any instance of a relation scheme obeying the dependencies specified above. The negative detection scheme $Scheme_{ND}$ would be most appropriate for membership queries, while $Scheme_{PC}$ would be appropriate for queries requiring enumeration of the member tuples. Conversely, known results from database theory regarding minimal representations translate to lower bounds on time and space requirements for these detection schemes.

C. Dynamic Samples

As mentioned in Section III, we expect our samples—and hence our distribution of the next packet generated by G—to change with time. Some factors contributing to this include:

1) If $x(t)$ is not in $Q(sample(t))$, it may become part of $sample(t+1)$ and $Q(sample(t+1))$ then properly contains $Q(sample(t))$ and, consequently, the distribution of the string generated at time $t+1$ differs from that at t. This assumes we have a mechanism for deciding $x(t)$ is generated by G, despite the fact that we flag it as a low probability string (e.g., it passes some investigative step).
2) We might want to delete packets from the sample that have not occurred recently. This might or might not alter $Q(sample(t))$, and hence, the distribution of the string generated at time $t+1$.

Extending our analytical treatment to include dynamically changing samples raises a number of questions. In principle, when a new sample is created at time t, a new set of detectors could be generated *de novo*. However, this is likely to be extremely inefficient. An area of future investigation is to analyze the techniques used to update the detector set in response to the specific types of perturbations in the sample.

D. Distance Measures

If we can define a metric over a match rule, then our detection schemes can be rewritten in terms of a distance measure. For instance using r-chunks:

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

where $d(x,S)$ is the number of windows that are present in string x but not in any string in S. We are interested in exploring schemes that are intermediate between conjunctive and disjunctive detection (both positive or negative). In these intermediate schemes, a string may be treated as self even if not all its window patterns have been observed before, i.e. $Scheme_M = \{x|d(x,S) \le \tau\}$ where a τ is some threshold. Intuitively, strings that differ a small amount from the sample are more likely to be a part of RS that those that differ a lot.

Extending this idea one step further, one can imagine distance measures that do not assign a uniform value to every match, but instead take into account structural and statistical properties of the sample in order to weigh the relative merits of distinct matches.

E. Implementation Issues

Although we have emphasized the representational power of different detection schemes in this paper, there are important implementation considerations that affect the choice of a match rule and detection scheme. Here, we briefly discuss the r-chunks match rule and contrast it with rcb, although, as shown in Section IV-C, the class of languages recognized by them are distinct. Nevertheless, the comparison highlights some important properties of r-chunks. We have so far described our detection schemes as a set of detectors, first because we find it useful for visualizing and analyzing their properties, and second because we consider that some applications might benefit from distributing them throughout several nodes; thus we make no assumptions as to their location. Nevertheless, it is important to note that there is a large body of work [69] regarding string matching that suggests a more efficient representation of detectors along with the algorithms that operate on them, but impose additional restrictions on the ease of distributing them. With this in mind we consider three implementation issues: The cost of generating detectors, the cost of storing detectors, and the cost of locating detectors.

1) Detector Generation: Given a self set S, a straightforward method for generating the appropriate r-chunks detectors is to search the entire sample to determine which patterns are present in each window, requiring $O(t|S|)$ time and $O(t2^r)$ space. Note that the detector set can be sorted during its generation at no extra cost (using bin sort and assuming $|S| > 2^r$). There have been several algorithms proposed for generating detectors for the rcb match rule [1], [3], [45], [46] that report linear generation time and $O(t^2 2^r)$ extra space.

2) Detector Storage: Intuitively, r-chunks requires more space than rcb since every rcb detector (l bits each) can be decomposed into t r-chunks detectors (tr bits). Nevertheless, depending on the particular structure of S, there is likely to be a significant number of repeated patterns in each window amongst the set of rcb detectors. Because r-chunks represents each distinct pattern once, this could lead to smaller space requirements in some cases. In general, the space required for r-chunks is $O(tr2^r)$.

3) Detector Location: Having detectors specific for each window (as in r-chunks) allows them to be stored in sorted order for each window. Thus, checking to see if a string belongs to self or not requires $O(tr)$ time while inserting or deleting a detector is just $O(r)$. There is also a potential that hashing schemes could reduce the search time.

F. The Crossover Closure and Genetic Algorithms

There is a tantalizing connection between the crossover closure described in this paper and the crossover operator often used in genetic algorithms. The crossover closure contains only a subset of all possible one-point crossovers in the self set, so the generalization achieved by r-contiguous bits is not identical with the space of coordinate hyperplanes defined by the crossover operator. Likewise, the r-chunks matching rule is nothing more than a restricted form of the schema notation often used to explain genetic algorithm behavior. An additional area of future investigation is to make these connections more precise and to determine if the Schema Theorem from genetic algorithms bears any relation to the processing performed by either of the contiguous bits match rules.

VIII. Conclusion

In this paper we presented a formal framework for analyzing different positive and negative detection schemes in the context of approximate matching. Although the framework we presented is quite general, the primary application we have in mind is anomaly intrusion detection, for example, detecting anomalous TCP connections in a local area network or detecting anomalous patterns of system calls in executing processes. We gave examples of how different partial matching rules fit into our scheme, including Hamming Distance, r-contiguous bits, and n-grams. We characterized the generalization induced by r-contiguous bits (and its relative r-chunks) by defining the crossover closure of a sample of l-length strings. Next, we showed that the crossover closure is related to the concept of a lossless join in relational database theory.

With this formal apparatus in place, we were then able to give some theoretical results on the relative power of the different detection schemes and matching rules. In particular, we showed that the r-chunks match rule subsumes r-contiguous bits matching; that (under rcb or r-chunks) the class of languages recognized by Negative Disjunctive Detection is the same as that of Positive Conjunctive Detection; that the crossover closure of a sample S exactly characterizes the class of languages recognized by Negative Disjunctive Detection and Positive Conjunctive Detection (under r-chunks matching).

Next, we considered the number of detectors that are required to provide maximal discrimination for fixed r under $Scheme_{PC}$ and $Scheme_{ND}$. We gave closed-form expressions for both schemes, allowing us for the first time to estimate how large a self set must be before negative detection is a computationally advantageous strategy. We then discussed the expected size of the crossover closure and explored how the use of permutations of the representation can reduce the size of the crossover closure. Finally, we showed that the class of languages recognized by r-contiguous bits, augmented with permutation masks, properly contains that recognized by Hamming Distance, and we showed that even with the use of permutations, there are some languages that cannot be recognized using r-contiguous bits matching.

The theoretical results presented here are significant because they address a large and quickly expanding body of experimental work in intrusion detection which use one of the detection schemes. It is our hope that these theoretical results will make it easier to understand the experimental results and to predict which approaches are most promising for which problems.

IX. Acknowledgments

The authors gratefully acknowledge the support of the National Science Foundation (grants ANIR-9986555 and DBI-0309147), 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. We would also like to thank John Holland, Andy Tyrrell, William Langdon and the anonymous reviewers who's comments greatly improved the presentation of this work. F.E. thanks Consejo Nacional de Ciencia y Tecnología (México) grant No. 116691/131686 for its financial support.

References

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

[2] S. Forrest, A. S. Perelson, L. Allen, and R. C. Kuri, "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, 1994.

[3] 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.

[4] J. W. Kappler, N. Roehm, and P. Marrack, "T-cell tolerance by clonal elimination in the thymus," *Cell*, vol. 49, pp. 273–280, April 24 1987.

[5] M. J. Kearns and U. V. Vazirani, *An Introduction to Computational Learning Theory*. Cambridge, MA: MIT Press, 1994.

[6] B. Scholköpf and A. Smola, *Learning with Kernels*. MIT Press, 2002.

[7] D. Angulin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, pp. 87–106, 1987.

[8] J. C. Schlimmer, "Concept acquisition through representational adjustment," Ph.D. dissertation, University of California, Irvine, 1987.

[9] T. Lane, "Machine learning techniques for the computer security domain of anomaly detection," Ph.D. dissertation, Purdue University, W. Lafayette, IN, Aug. 2000.

[10] G. B. Lamont, R. E. Marmelstein, and D. A. V. Veldhuizen, "A distributed architecture for a self-adaptive computer virus immune system," in *New Ideas in Optimization*, ser. Advanced Topics in Computer Science Series. London: McGraw-Hill, 1999, pp. 167–183.

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

[12] S. Hofmeyr and S. Forrest, "Architecture for an artificial immune system," *Evolutionary Computation Journal*, vol. 8, no. 4, pp. 443–473, 2000.

[13] 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)*. San Francisco, CA: Morgan-Kauffman, 2001, pp. 1330–1337.

[14] 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 *Fourth International Symposium, Recent Advances in Intrusion Detection*, W. Lee, L. Me, and A. Wespi, Eds. Berlin: Springer, 2001, pp. 117–133.

[15] D. Dasgupta and S. Forrest, "Novelty detection in time series data using ideas from immunology," in *Proceedings of the International Conference on Intelligent Systems*, 1996, best paper award.

[16] 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.

[17] D. L. Chao and S. Forrest, "An aesthetic immune system (tentative title)," in *Artificial Life VIII: The 8th International Conference on the Simulation and Synthesis of Living Systems*, Submitted May, 2002.

[18] S. Hofmeyr, "An immunological model of distributed detection and its application to computer security," Ph.D. dissertation, University of New Mexico, Albuquerque, NM, 1999.

[19] V. Barnett and T. Lewis, *Outliers in Statistical Data*. John Wiley and Sons, 1994.

[20] E. Eskin, "Anomaly detection over noisy data using learned probability distributions," in *Proceedings 17th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 2000, pp. 255–262.

[21] D. Freedman, R. Psani, and R. Purves, *Statistics*. New York: W.W. Norton, 1978.

[22] D. Hoaglin, F. Mosteller, and J. Tukey, *Understanding Robust and Exploratory Data Analysis*. New York: John Wiley and Sons, 1983.

[23] D. H. Fisher, “Knowledge acquisition via incremental conceptual clustering,” in *Machine Learning 2(2)*, 1987, pp. 139–172.

[24] S. J. Hanson and M. Bauer, “Conceptual clustering, categorization, and polymorphy,” in *Machine Learning, 3(4)*, 1989, pp. 343–372.

[25] A. Jain and R. Dubes, *Algorithms for Clustering Data*. Prentice Hall, 1988.

[26] L. Kaufman and P. Rousseeuw, *Finding Groups in Data*. John Wiley and Sons, 1990.

[27] E. Knorr and R. Ng, “Algorithms for mining distance based outliers in large databases,” in *Proceedings 24th VLDB*, 1998, pp. 392–403.

[28] R. Ng and J. Han, “Efficient and effective clustering methods for spacial datamining,” in *Proceedings 20th VLDB*, 1994, pp. 144–155.

[29] G. Salton and M. J. McGill, *Introduction to Modern Retrieval*. McGraw-Hill, 1983.

[30] A. Arning, R. Agrawal, and P. Raghavan, “A linear method for deviation detection in large databases,” in *2nd Knowledge Discovery and Data Mining*, 1996, pp. 164–169.

[31] E. Knorr and R. Ng, “A unified approach for mining: Properties and computation,” in *Proceedings 3rd Knowledge Discovery and Data Mining*, 1997, pp. 219–222.

[32] P. Helman and R. Gore, “Prioritizing information for the discovery of phenomena,” *Journal of Intelligent Information Systems*, vol. 11, no. 2, pp. 99–138, September/October 1998.

[33] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*. Addison-Wesley, 1974.

[34] D. E. Denning, “An intrusion-detection model,” *IEEE Transactions on Software Engineering*, vol. 13, no. 2, p. 222, February 1987.

[35] H. Teng, K. Chen, and S. Lu, “Adaptive real-time anomaly detection using inductively generated sequential patterns,” in *Proceedings of the IEEE Symposium on Research in Computer Security and Privacy*, IEEE. Los Alamitos, CA: IEEE Computer Society Press, 1990.

[36] H. S. Javitz and A. Valdes, “The SRI IDES statistical anomaly detector,” in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1991, pp. 316–326.

[37] H. S. Javitz, A. Valdes, T. F. Lunt, A. Tamaru, M. Tyson, and J. Lowrance, “Next generation intrusion detection expert system (NIDES),” SRI International, Menlo Park, CA, Tech. Rep. A016, 1993.

[38] P. Helman and G. Liepins, “Statistical foundations of audit trail analysis for the detection of computer misuse,” *IEEE Transactions on Software Engineering*, vol. 19, no. 9, pp. 886–901, September 1993.

[39] P. Helman, G. Liepins, and W. Richards, “Foundations of intrusion detection,” in *Proceedings of the Fifth Computer Security Foundations Workshop*, Franconia, NH, 1992, pp. 114–120.

[40] M. Burgess, H. Haugerud, and S. Straumsnes, “Measuring system normality i: Scales and characteristics,” 2000.

[41] D. Q. Naiman, “Statistical anomaly detection via httpd data analysis,” *Computational Statistics and Data Analysis*, in press.

[42] J. K. Percus, O. Percus, and A. S. Perelson, “Probability of self-nonself discrimination,” in *Theoretical and Experimental Insights into Immunology*, A. S. Perelson and G. Weisbuch, Eds. NY: Springer-Verlag, 1992.

[43] ——, “Predicting the size of the antibody combining region from consideration of efficient self/non-self discrimination,” *Proceedings of the National Academy of Science*, vol. 90, pp. 1691–1695, 1993.

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

[45] ——, “Discriminative power of the receptors activated by k-contiguous bits rule,” *Journal of Computer Science and Technology*, vol. 1, no. 3, pp. 1–13, 2000.

[46] ——, “Deriving concise description of non-self patterns in an artificial immune system,” in *New Learning Paradigm in Soft Computing*, S. T. Wierzchon, L. C. Jain, and J. Kacprzyk, Eds. Heidelberg New York: Physica-Verlag, 2001, pp. 438–458.

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

[48] J. Balthrop, S. Forrest, and M. Glickman, “Revisting lisys: Parameters and normal behavior,” in *Proceedings of the 2002 Congress on Evolutionary Computation*, in press.

[49] D. Dasgupta and F. Gonzalez, “An immunity-based technique to characterize intrusions in computer networks,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, June 2002.

[50] L. Laboratories, “Darpa intrusion detection evaluation,” http://www.ll.mit.edu/IST/ideval/index.html, 1999.

[51] D. W. Bradley and A. M. Tyrrell, “The architecture for a hardware immune system,” in *The Third NASA/DoD workshop on Evolvable Hardware*, D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, Eds. Long Beach, California: IEEE Computer Society, 12-14 July 2001, pp. 193–200.

[52] ——, “A hardware immune system for benchmark state machine error detection,” in *proceedings of the Congress on Evolutionary Computation 2002 (CEC2002)*, Honolulu, Hawaii, May 2002.

[53] ——, “Immunotronics: Novel finite state machine architectures with built in self test using self-nonself differentiation,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 227–238, June 2002.

[54] M. Roesch, “Snort,” http://www.snort.org/.

[55] S. Hofmeyr, A. Somayaji, and S. Forrest, “Intrusion detection using sequences of system calls,” *Journal of Computer Security*, vol. 6, pp. 151–180, 1998.

[56] D. J. Smith, S. Forrest, D. H. Ackley, and A. S. Perelson, “Variable efficacy of repeated annual influenza vaccination,” *Proceedings of the National Academy of Sciences (PNAS)*, vol. 96, pp. 14 001–14 006, 1999.

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

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

[59] P. F. Brown, V. J. D. Pietra, P. V. de Souza, J. C. Lai, and R. L. Mercer, “Class-based n-gram models of natural language,” in *Proceedings of the IBM Natural Language ITL*, Paris, FR, 1990.

[60] M. Damashek, “Gauging similarity with n-grams: Language-independent categorization of text,” *Science*, vol. 267, pp. 843–848, 1995.

[61] W. Lee and S. Stolfo, “Data mining approaches for intrusion detection,” in *7th USENIX Security Symposium*, 1998.

[62] C. Marceau, “Characterizing the behavior of a program using multiple-length n-grams,” in *Proceedings of the New Security Paradigms Workshop*, Cork, Ireland, 2000.

[63] K. Tan and R. Maxion, “”why 6?” defining the operational limits of stide, and anomaly-based intrusion detector,” in *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland, CA: IEEE Press, 2002.

[64] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff, “A sense of self for unix processes,” in *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*. IEEE Press, 1996.

[65] A. Somayaji and S. Forrest, “Automated response using system-call delays,” in *Usenix Security Syposium*, 2000.

[66] C. Marceau, “Characterizing the behavior of a program using multiple-length N-grams,” in *Proceedings of the 2000 new security paradigm workshop*, Sept. 2000, pp. 101–110. [Online]. Available: ftp://ftp.oracorp.com/documents/MultiLengthStrings.pdf

[67] P. Helman and J. Bhangoo, “A statistically based system for prioritizing information exploration under uncertainty,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 27, no. 4, pp. 449–466, July 1997.

[68] J. Balthrop, F. Esponda, S. Forrest, and M. Glickman, “Coverage and generalization in an artificial immune system,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-02)*, in press.

[69] D. Gusfield, *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, 1997.

Fernando Esponda received his degree in computer engineering from the Instituto Tecnológico Autónomo de México (ITAM), Mexico City, 1995, and studied a masters in the Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas (IIMAS-UNAM), Mexico City. He is currently working towards his Ph.D. degree at the University of New Mexico. His research interests are in biologically inspired adaptive systems and Machine Learning.

Stephanie Forrest is Professor of Computer Science at the University of New Mexico in Albuquerque. Professor Forrest received her Ph.D. in Computer and Communication Sciences from the University of Michigan. Before joining UNM she worked for Teknowledge Inc. and was a Director's Fellow at the Center for Nonlinear Studies, Los Alamos National Laboratory. She is currently a member of the Santa Fe Institute science board and external faculty, and served as SFI's Interim Vice President 1999-2000. Her research interests are in adaptive systems, including genetic algorithms, computational immunology, biological modeling, and computer security.

Paul Helman was born in Brooklyn, NY, in 1954. He received the BA from Dickinson College in 1976, the MS from Stanford University in 1977, and the Ph.D. from the University of Michigan in 1982. Currently, he is Professor of Computer Science at the University of New Mexico. Dr. Helman's research interest include bioinformatics, machine learning, data mining, database theory, and combinatorial optimization. He also has authored two computer science text books, Walls and Mirrors, and The Science of Database Management.

NOTES

[1]Mature T-cells have survived at least two other censoring operations, one involving genetic rearrangements and one involving positive selection.

[2]The n-gram method described here is slightly different from the lookahead pairs method described in [64], [65].

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

[4]The labels for window numbers could be recovered from the level by using a topological sort.

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

LIST OF FIGURES