Negative Selection Algorithms on Strings with Efficient Training and Linear-Time Classification

Michael Elberfeld, Johannes Textor

Institut für Theoretische Informatik, Universität zu Lübeck, 23538 Lübeck, Germany

Abstract

A string-based negative selection algorithm is an immune-inspired classifier that infers a partitioning of a string space Σ^ℓ into "normal" and "anomalous" partitions from a training set S containing only samples from the "normal" partition. The algorithm generates a set of patterns, called "detectors", to cover regions of the string space containing none of the training samples. Strings that match at least one of these detectors are then classified as "anomalous". A major problem with existing implementations of this approach is that the detector generating step needs exponential time in the worst case. Here we show that for the two most widely used kinds of detectors, the r-chunk and r-contiguous detectors based on partial matching to substrings of length r, negative selection can be implemented more efficiently by avoiding generating detectors altogether: For each detector type, training set $S \subseteq \Sigma^\ell$ and parameter $r \leq \ell$ one can construct an automaton whose acceptance behaviour is equivalent to the algorithm's classification outcome. The resulting runtime is $O(|S|\ell r|\Sigma|)$ for constructing the automaton in the training phase and $O(\ell)$ for classifying a string.

Key words: negative selection, r-chunk detectors, r-contiguous detectors, artificial immune systems, anomaly detection

1. Introduction

The adaptive immune system successfully protects vertebrate species, including us humans, from being extinguished by pathogens. Remarkably, the immune system accomplishes this without "knowing" what a pathogen is. Instead, it tolerates the tissues, cells and molecules that are normal components of its host organism – the *self* – and simply attacks everything else – the *nonself*. While this paradigm is not perfectly accurate – nonself does include most dangerous things like viruses, bacteria and fungi, but also benign ones like a donated organ – it apparently works quite well. Self-nonself-discrimination is thus a natural source of inspiration for computer security: Computer systems and networks are frequently attacked by viruses, worms and other malware, and a computer program that discriminates with perfect accuracy between benign and malign software cannot exist. As an approximation, could the immune system's "nice hack" [1] be transferred to the computer security domain?

A popular approach to design such computer immune systems is inspired by the way that the real immune system generates *T cells* with the ability to detect nonself entities. This process is known as *negative selection* [2, 3]: The receptors of newborn T cells are assembled from randomly combinated gene fragments. In an organ called the *thymus*, the T cells are then exposed to proteins from self, and cells whose receptors match such a self protein are bound to die. Only those that survive negative selection may leave the thymus, and use their receptors to screen the organism for nonself proteins. An algorithmic abstraction of this process is called a *negative selection algorithm*.

The negative selection algorithms that we consider in this paper are binary classifiers operating on a string space Σ^ℓ. The classification problem is posed as follows (Figure 1): Σ^ℓ is assumed to be pre-partitioned in two pairwise disjoint subsets $\mathcal{S}$ (self) and $\mathcal{N}$ (nonself). The strings can represent, for example, data packets in a computer network [4] or sequences of system calls from UNIX processes [5], where the self and nonself partitions would correspond

Email addresses: elberfeld@tcs.uni-luebeck.de (Michael Elberfeld), textor@tcs.uni-luebeck.de (Johannes Textor)

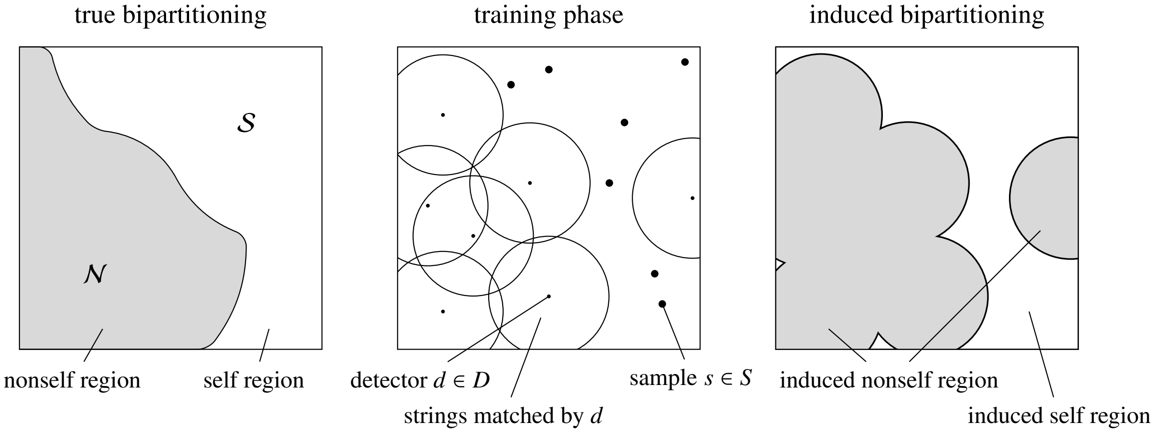


Figure 1: The classification problem that is solved by a negative selection algorithm. The string universe Σ^ℓ is prepartitioned in two regions $\mathcal{S}$ (self) and $\mathcal{N}$ (nonself). The classifier is given a training set $S \subseteq \mathcal{S}$ (large dots) and generates a detector set D (small dots) to cover regions of the universe containing none of the training examples (circles). The detector set induces a classification boundary that approximates the partitioning of Σ^ℓ into $\mathcal{S}$ and $\mathcal{N}$.

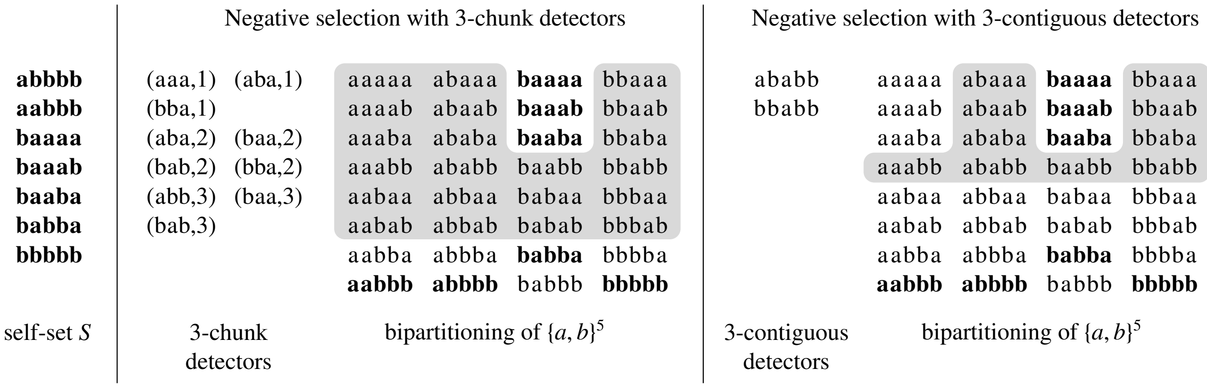
to "normal" and "anomalous" behaviour, respectively. The algorithm is given a sample $S \subseteq \mathcal{S}$ of self strings, called *self-set*, and a set $M \subseteq \Sigma^\ell$ of strings to classify, called *monitor set*. It then generates a set D of patterns called *detectors*. In analogy to the T cells in the immune system, this is typically done by generating the detectors randomly and discarding those that match any string in the self-set. Consequently, each string $m \in M$ is classified by labeling m as non-self if it is matched by any detector, and self otherwise. In particular, m is never labeled non-self if it also occurs in the self-set.

From a broader machine learning perspective, negative selection is usually described as an *anomaly detection* technique [6, 7]. The following two important properties distinguish negative selection from many well-known classifiers: (1) The training data consists of examples from only one class. Other techniques with this property include classifiers based on kernel density estimation [8, 9] and the one-class support vector machine [10]. (2) Classification is based on a *negative representation* of training data, typically on short substrings (r-grams) that do not occur in the self-set. While positive representations such as the r-gram frequency distribution used e.g. for identification of language [11] and text categorization [12] are more common in the machine learning domain, similar negative representations have been studied in string theory. For example, certain sets of non-occurring substrings *(forbidden words)* can be used to describe the complexity of a language [13].

1.1. Contribution of this paper

This paper presents two algorithms that implement string-based negative selection with r-chunk and r-contiguous detectors by generating compressed representations of the respective detector sets, from which automata are constructed that simulate the classification outcome through their acceptance behaviour. Both algorithms use time $O(|S|\ell r|\Sigma|)$ to construct an automaton for a given self-set S and parameter r, which is equivalent to the *training phase* of the simulated negative selection algorithm. The automaton can be used to classify each string in linear time $O(\ell)$. This improves upon the exponential worst-case complexity of existing algorithms, and thus removes one major obstacle for applying negative selection to real-world problems [14, 15, 16]. In comparison to our preliminary conference version [17], the algorithms presented in this paper are based on prefix trees instead of patterns. This reduces the overall runtime significantly (Table 1), generalizes to higher alphabets, and allows for a simpler and more concise presentation. In addition to the classification itself, the automata can also be used to efficiently count the detectors and, if necessary, enumerate them explicitly.

The r-chunk and r-contiguous detectors considered here are among the most common ones in the artificial immune systems literature [6]: (1) An *r-contiguous detector* is a string of length ℓ and matches all strings to which it is identical in at least r contiguous positions. (2) An *r-chunk detector* is a string of length r (or r-gram) with a position index and matches all strings in which the r-gram occurs at that position. Figure 2 shows an example self-set $S \subseteq \{a, b\}^5$ along with the complete sets of 3-chunk and 3-contiguous detectors that do not match any string in S, as well as the partitioning of $\{a, b\}^5$ induced by these detector sets. The r-contiguous detectors are directly based on a model of antigen recognition by T cell receptors [18, 2], and r-chunk detectors were later introduced to achieve better results on data where adjacent regions of the input strings are not necessarily semantically correlated, such as network data packets [4].



	Negative selection with 3-chunk detectors					Negative selection with 3-contiguous detectors				
abbbb	(aaa,1) (aba,1)	aaaaa	abaaa	**baaaa**	bbaaa	ababb	aaaaa	abaaa	**baaaa**	bbaaa
aabbb	(bba,1)	aaaab	abaab	**baaab**	bbaab	bbabb	aaaab	abaab	**baaab**	bbaab
baaaa	(aba,2) (baa,2)	aaaba	ababa	**baaba**	bbaba		aaaba	ababa	**baaba**	bbaba
baaab	(bab,2) (bba,2)	aaabb	ababb	baabb	bbabb		aaabb	ababb	baabb	bbabb
baaba	(abb,3) (baa,3)	aabaa	abbaa	babaa	bbbaa		aabaa	abbaa	babaa	bbbaa
babba	(bab,3)	aabab	abbab	babab	bbbab		aabab	abbab	babab	bbbab
bbbbb		aabba	abbba	**babba**	bbbba		aabba	abbba	**babba**	bbbba
		aabbb	**abbbb**	babbb	**bbbbb**		**aabbb**	**abbbb**	babbb	**bbbbb**
self-set S	3-chunk detectors	bipartitioning of $\{a, b\}^5$				3-contiguous detectors	bipartitioning of $\{a, b\}^5$			

Figure 2: An example self-set $S \subseteq \{a, b\}^5$ along with all 3-chunk detectors and 3-contiguous detectors that do not match any string in S is shown. For both detector types, the induced bipartitionings of the shape space $\{a, b\}^5$ are illustrated with strings that are classified as nonself having a gray background and strings that are classified as self having a white background. Bold strings are members of the self-set. The *generalization region* of the negative selection classifier consists of the strings that are classified as self but do not occur in the self-set. These strings are also called "holes" in the negative selection literature [6, 19].

1.2. Related Work on String-Based Negative Selection

The question whether negative selection with r-contiguous and r-chunk detectors can be implemented in polynomial worst-case time was open for several years. The complexity issues caused by the verbatim abstraction of negative selection as performed by the immune system are two-fold: On one hand, if the self partition is only a small fraction of Σ^ℓ, then there is an exponential number of potential detectors, and it is unclear how many of these have to be generated to achieve an acceptable detection rate. The early work of D'haeseleer and others [20, 21] addressed these problems by proving lower bounds on the number of required detectors, and presenting algorithms that generate detectors by a structured exhaustive search. However, these algorithms still have a runtime exponential in r. Similar algorithms and heuristics were later proposed by Wierzchoń [22], Ayara et al. [23], and Stibor et al. [24]. In an effort to clarify the computational complexity of negative selection, Stibor and coworkers studied the associated decision problem [25, 19]: Given a self-set S, can an r-contiguous detector be generated that does not match any string in S? It was suspected that this decision problem might be NP-complete [14], although no completeness proof had been shown. These ongoing difficulties led some in the field to conclude that negative selection is computationally too expensive for real-world datasets [15, 26]. This issue was settled by the preliminary conference version of the present paper, which demonstrated for the first time that string-based negative selection is feasible in polynomial time [17]. Most recently, Liśkiewicz and Textor discussed the idea of negative selection without explicit detector generation from a learning theoretical perspective [27].

1.3. Organization of This Paper

We start out by defining the formal underpinnings of our algorithms in the upcoming section. Afterwards, in Section 3, we sketch the construction of an automaton consisting of prefix trees and failure links that can be used to simulate negative selection with r-chunk detectors. This rather straightforward construction is used as a basis for the more involved one in Section 4, where we transform the automaton into one that allows linear-time classification with respect to r-contiguous detectors.

r-chunk detector-	asymptotic runtime	
based algorithms	training phase	classification phase
Stibor et al. [24]	$(2^r + \|S\|)(\ell - r + 1)$	$\|D\|\ell$
Elberfeld, Textor [17]	$\|S\|(\ell - r + 1)r^2$	$\|S\|\ell^2 r$
Present paper	$\|S\|\ell r$	ℓ

r-contiguous detector-	asymptotic runtime	
based algorithms	training phase	classification phase
D'haeseleer et al. [20] (linear)	$(2^r + \|S\|)(\ell - r)$	$\|D\|\ell$
D'haeseleer et al. [20] (greedy)	$2^r\|S\|(\ell - r)$	$\|D\|\ell$
Wierzchón [22]	$2^r(\|D\|(\ell - r) + \|S\|)$	$\|D\|\ell$
Elberfeld, Textor [17]	$\|S\|^3\ell^3 r^3$	$\|S\|^2\ell^3 r^3$
Present paper	$\|S\|\ell r$	ℓ

Table 1: Comparison of our results with the runtimes of previously published algorithms. All runtimes are given for a binary alphabet ($|\Sigma| = 2$) since not all algorithms are applicable to arbitrary alphabets. The parameter $|D|$, the desired number of detectors, is only applicable to algorithms that generate detectors explicitly – our algorithms produce the results that would be obtained with the maximal number of generated detectors.

2. Preliminaries

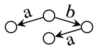
In this section, we define the formal background of our work. First we review some basic terms related to strings and pattern matching techniques like automata. Then we define r-chunk detectors, r-contiguous detectors, and the corresponding classification approaches.

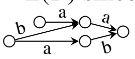
2.1. Strings, Substrings and Languages

An *alphabet* Σ is a nonempty and finite set of *symbols*. A *string* $s \in \Sigma^*$ is a sequence of symbols from Σ, and its length is denoted by $|s|$. The string with $|s| = 0$ is called the *empty string*. Given an index $i \in \{1, \ldots, |s|\}$, then $s[i]$ is the symbol at position i in s. Given two indices i and j, whenever $j \geq i$, then $s[i \ldots j]$ is the *substring* of s with length $j - i + 1$ that starts at position i and whenever $j < i$, then $s[i \ldots j]$ is the empty string. If $i = 1$, then $s[i \ldots j]$ is a *prefix* of s and, if $j = |s|$, then $s[i \ldots j]$ is a *suffix* of s. For a *proper* prefix or suffix s' of s, we have in addition $|s'| < |s|$. Given a string $s \in \Sigma^\ell$, another string $d \in \Sigma^r$ with $1 \leq r \leq \ell$, and an index $i \in \{1, \ldots, \ell - r + 1\}$, we say that *$d$ occurs in s at position i* if $s[i \ldots i + r - 1] = d$.

A set of strings $S \subseteq \Sigma^*$ is called a *language*. For two indices i and j, we define $S[i \ldots j] = \{s[i \ldots j] \mid s \in S\}$. We say that *$S$ avoids a string d at position i* if d occurs in no $s \in S$ at position i. Alternatively, we say that *S avoids (d, i)*.

2.2. Prefix Trees, Prefix DAGs, and Automata

A *prefix tree* T such as  is a rooted directed tree with edge labels from Σ where for all $\sigma \in \Sigma$, every node has at most one outgoing edge labeled with σ. For a string s, we write $s \in T$ if there is a path from the root of T to a leaf such that s is the concatenation of the labels on this path. The language $L(T)$ described by T is defined as the set of all strings that have a nonempty prefix $s \in T$. For example, for T above we have $a \in T$ and $ba \in T$, but $b \notin T$. Furthermore, $a \in L(T)$, $ab \in L(T)$ since $a \in T$ and $bb \notin L(T)$ since no prefix of bb lies in T.

A *prefix* DAG D such as  is a directed acyclic graph with edge labels from Σ, where again for all $\sigma \in \Sigma$, every node has at most one outgoing edge labeled with σ. In analogy to prefix trees, we will use the terms root and leaf to refer to a node without incoming and outgoing edges, respectively. We write $s \in D$ if there is a root node n_r and a leaf node n_l in D with a path from n_r to n_l that is labeled by s. Given $n \in D$, the language $L(D, n)$ contains all strings that have a nonempty prefix that labels a path from n to some leaf. For instance, if D is the DAG above and n is its upper left node, then $L(D, n)$ consists of all strings starting with aa. Moreover, we define $L(D) = \bigcup_{n \text{ is a root of } D} L(D, n)$.

We will construct finite automata to decide the membership of strings in languages. Formally, a *finite automaton* is a tuple $M = (Q, q_\text{i}, Q_\text{a}, \Sigma, \Delta)$, where Q is a *set of states* with a distinguished *initial state* $q_\text{i} \in Q$, $Q_\text{a} \subseteq Q$ the set of *accepting states*, Σ the *alphabet* of M, and $\Delta \subseteq Q \times \Sigma \times Q$ the *transition relation*. Furthermore, we assume that the transition relation is *unambiguous*: for every $q \in Q$ and every $\sigma \in \Sigma$ there is at most one $q' \in Q$ with $(q, \sigma, q') \in \Delta$. It is common to represent the transition relation as a graph with node set Q (with the initial state and the accepting states highlighted properly) and labeled edges (a σ-labeled edge from q to q' if $(q, \sigma, q') \in \Delta$.) An automaton M is said to *accept* a string s if its graph contains a path from q_{i} to some $q \in Q_{\mathrm{a}}$ whose concatenated edge labels equal s (note that this path may contain loops). The language $L(M)$ contains all strings accepted by M. Note that every prefix DAG D can be turned into a finite automaton M with $L(D) = L(M)$. For a more detailed discussion of automata-based string processing, we refer to the textbook of Crochemore, Hancart and Lecroq [28].

2.3. Detectors and Self-Nonself-Discrimination

We fix an alphabet Σ, a string length ℓ, a *self-set* $S \subseteq \Sigma^{\ell}$, and a matching parameter $r \in \{1, \ldots, \ell\}$.

Definition 2.1 (r-chunk detector). An *r-chunk detector* (d, i) is a tuple of a string $d \in \Sigma^r$ and an index $i \in \{1, \ldots, \ell - r + 1\}$. It *matches* a string s if d occurs in s at position i.

The *set of r-chunk detectors for S*, denoted by CHUNK(S, r), contains exactly the r-chunk detectors (d, i) that do *not* match any string in S. Let $m \in \Sigma^{\ell}$. The string m is *nonself with respect to S and r-chunk detectors* if m matches an r-chunk detector from CHUNK(S, r) and *self*, otherwise. The set CHUNK-NONSELF(S, r) contains exactly the strings of length ℓ over Σ that are nonself with respect to S and r-chunk detectors.

Definition 2.2 (r-contiguous detector). An *r-contiguous detector* is a string $d \in \Sigma^{\ell}$. It *matches* a string $s \in \Sigma^{\ell}$ if there is an index $i \in \{1, \ldots, \ell - r + 1\}$ where $d[i \ldots i + r - 1]$ occurs in s.

Similar to the chunk detector case, we define the *set of r-contiguous detectors for S and r*, CONT(S, r), as the set of all r-contiguous detectors that do not match any string in S. Let $m \in \Sigma^{\ell}$. The string m is *nonself with respect to S and r-contiguous detectors* if m matches an r-contiguous detector from CONT(S, r) and *self*, otherwise. The set CONT-NONSELF(S, r) contains exactly the strings that are nonself with respect to S and r-contiguous detectors.

Figure 2 from the introduction shows an example of a self-set $S \subseteq \{a, b\}^5$, the corresponding detector sets CHUNK$(S, 3)$ and CONT$(S, 3)$, and the corresponding induced partitions of the shape space into self and nonself.

3. Negative Selection with Chunk Detectors

In this section, we discuss how to construct automata for CHUNK-NONSELF(S, r). The construction is a combination of two standard string processing tools: prefix trees and failure links. It will be a building block for the more intricate construction in the next section.

Theorem 3.1. *There exists an algorithm that, given any $S \subseteq \Sigma^{\ell}$ and $r \in \{1, \ldots, \ell\}$, constructs a finite automaton M with $L(M) \cap \Sigma^{\ell} =$ CHUNK-NONSELF(S, r) in time $O(|S|\ell r|\Sigma|)$.*

PROOF. First let us discuss how we can classify m in time $O(\ell r)$ using prefix trees. By definition, a string $m \in \Sigma^{\ell}$ lies in the set CHUNK-NONSELF(S, r) exactly if S avoids $(m[i \ldots i + r - 1], i)$ for some index $i \in \{1, \ldots, \ell - r + 1\}$. Hence, if we construct for every position $i \in \{1, \ldots, \ell - r + 1\}$ a prefix tree T_i with $L(T_i) \cap \Sigma^r = \Sigma^r \setminus S[i \ldots i + r - 1]$, we can classify m in time $O(\ell r)$ by checking for each position independently if $m[i \ldots i + r - 1]$ is in $L(T_i)$. If this is the case for at least one index i, we classify m as nonself. Each prefix tree T_i can be constructed as follows: Start with an empty prefix tree and insert every $s \in S[i \ldots i + r - 1]$ into it. Next, for every non-leaf node n and every $\sigma \in \Sigma$ where no edge with label σ starts at n, create a new leaf n' and an edge (n, n') labeled with σ. Finally, delete every node from which none of the newly created leaves is reachable. For the resulting prefix tree we have $L(T_i) \cap \Sigma^r = \Sigma^r \setminus S[i \ldots i + r - 1]$.

To enable a classification in time $O(\ell)$, we construct an automaton from the prefix trees above by inserting failure links between the prefix trees of adjacent levels, similar to the well-known algorithm of Knuth, Morris and Pratt [29]. Briefly, the idea of our failure link method is as follows: If a mismatch occurs in a prefix tree T_i at a position k, then we need not restart from the root of tree T_{i+1}, but can go directly to the node in T_{i+1} that corresponds to the last $k - 1$ symbols read. By inserting the failure links from right to left, turning the prefix trees into a prefix DAG, we can inductively ensure that either such a node exists or there is no match at all.

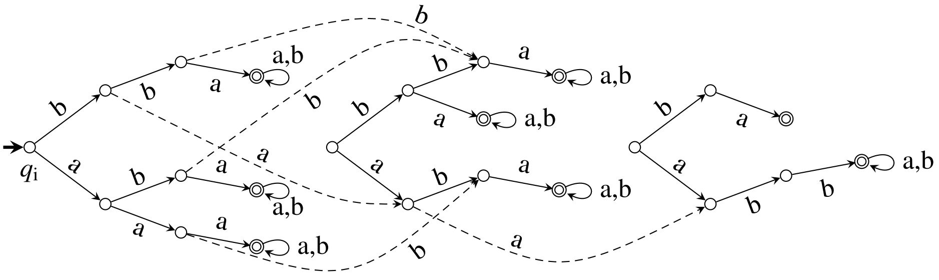


Figure 3: The constructed automaton M with $L(M) \cap \{a, b\}^5 =$ CHUNK-NONSELF$(S, 3)$ where S is the self-set from Figure 2. The solid lines are from the prefix trees T_i, the dashed lines are failure links. Note that M contains some states that are not reachable from the initial state q_i. These can be removed from M, but are shown here to illustrate the underlying prefix trees T_1, T_2, and T_3.

We start by letting D be the disjoint union of $T_1, \ldots, T_{\ell-r+1}$. Then we process the levels from $i = \ell - r$ down to 1 iteratively as follows: Consider every node n from T_i and every symbol $\sigma \in \Sigma$ where n has no outgoing edge with label σ. Let s be the string on the path from the root of T_i to n. Let $s' = s\sigma$ and, if it exists, let n' be the end node of the path from the root of T_{i+1} that is labeled by $s'[2 \ldots |s'|]$. In case that n' exists, we insert an edge from n to n' with label σ. Edges that are constructed in this step are called *failure links*. By induction one can show that after every iteration i we have $L(T_i) \cap \Sigma^{l-i+1} =$ CHUNK-NONSELF$(S[i \ldots \ell], r)$. Finally, we turn D into a finite automaton with the claimed property by making all leaves accepting states with self-loops for all $\sigma \in \Sigma$ and setting the initial state to the root of T_1. An example of this construction is shown in Figure 3.

Each prefix tree T_i can be constructed in time $O(|S|r|\Sigma|)$. The failure links between each pair of adjacent levels i and $i + 1$ can be inserted in time $O(|S|r|\Sigma|)$ by a simultaneous recursive traversal of T_i and T_{i+1}. Since the number of levels is $\ell - r + 1$, we obtain the claimed runtime. □

4. Negative Selection with Contiguous Detectors

In this section, we show how to efficiently construct automata for the languages CONT(S, r) and CONT-NONSELF(S, r), respectively. We first discuss the construction of an automaton for CONT(S, r), which will prove the following theorem:

Theorem 4.1. *There exists an algorithm that, given any $S \in \Sigma^\ell$ and $r \in \{1, \ldots, \ell\}$, constructs a finite automaton M with $L(M) \cap \Sigma^\ell =$ CONT(S, r) in time $O(|S|\ell r|\Sigma|)$.*

The construction in this section is more complex than the one in the previous section since, in order to accept CONT(S, r), it does not suffice to determine non-occurring length-r substrings for the levels independently. Instead, we need to determine non-occurring substrings that can be extended by non-occurring substrings from other levels to form strings of length ℓ – the r-contiguous detectors.

Let $S \subseteq \Sigma^\ell$, $d \in \Sigma^\ell$, $r \in \{1, \ldots, \ell\}$, and $(d', i) \in \Sigma^{\leq r} \times \{1, \ldots, \ell - r + 1\}$. The string d is an *(S, r)-avoiding right-completion of* (d', i) if (1) d' occurs in d at position i and S avoids (d', i), and (2) for all $j \in \{i + 1, \ldots, \ell - r + 1\}$, there is a string $d'' \in \Sigma^{\leq r}$ such that d'' occurs in d at position j and S avoids (d'', j). If property (2) is phrased with j ranging from 1 to $i - 1$, then d is an *(S, r)-avoiding left-completion of* (d', i). With this definition we have $d \in$ CONT(S, r) iff there exists $(d', i) \in \Sigma^{\leq r} \times \{1, \ldots, \ell - r + 1\}$ such that d is both an (S, r)-avoiding left-completion and an (S, r)-avoiding right-completion of (d', i).

To prove Theorem 4.1, we first prove the following Lemma:

Lemma 4.2. *There exists an algorithm that, given any $S \subseteq \Sigma^\ell$ and $r \in \{1, \ldots, \ell\}$, constructs a prefix* DAG *D with roots $\rho_1, \ldots, \rho_{\ell-r+1}$ such that $L(D, \rho_i) \cap \Sigma^{\ell-i+1} =$ CONT$(S, r)[i \ldots \ell]$ for every $i \in \{1, \ldots, \ell - r + 1\}$ in time $O(|S|\ell r|\Sigma|)$.*

Proof. The construction of D is done in four phases, presented and discussed in the next four paragraphs. While the following proof text explains the basic ideas and their correctness, the detailed computational steps are shown by the pseudocode in Figure 4.

```
Procedure CONSTRUCT-CONTIGUOUS-DETECTOR-DAG($S, r$)
Construct prefix trees:
    for $i = 1$ to $\ell - r + 1$ do
        $T_i \leftarrow$ prefix tree with $L(T_i) \cap \Sigma^r = \Sigma^r \setminus S[i \ldots i + r - 1]$
Trim the trees in a right-to-left pass:
    $T^{\mathrm{R}}_{\ell-r+1} \leftarrow T_{\ell-r+1}$
    for $i = \ell - r$ down to 1 do
        $T^{\mathrm{R}}_i \leftarrow$ empty prefix tree
        for each string $s \in T_i$ do
            if there exists $s' \in T^{\mathrm{R}}_{i+1}$ such that $s[2 \ldots |s|]$ is a prefix of $s'$ then insert $s$ into $T^{\mathrm{R}}_i$
Trim the trees in a left-to-right pass:
    $T^{\mathrm{L}}_1 \leftarrow T^{\mathrm{R}}_1$
    for $i = 2$ to $\ell - r + 1$ do
        $T^{\mathrm{L}}_i \leftarrow$ empty prefix tree
        for each string $s \in T^{\mathrm{R}}_i$ do
            if there exists $s' \in T^{\mathrm{L}}_{i-1}$ such that $s'[2 \ldots |s'|]$ is a prefix of $s$ then insert $s$ into $T^{\mathrm{L}}_i$
Weave the trees together into a prefix DAG:
    $D_{\ell-r+1} \leftarrow T^{\mathrm{L}}_{\ell-r+1}$
    for $i = \ell - r$ down to 1 do
        $D_i \leftarrow$ disjoint union of $D_{i+1}$ and $T^{\mathrm{L}}_i$;  $\rho_i \leftarrow$ root of $T^{\mathrm{L}}_i$
        for each string $s \in T^{\mathrm{L}}_i$ do
            $(n, \sigma, n') \leftarrow$ last labeled edge on the $s$-path from $\rho_i$ in $T^{\mathrm{L}}_i$
            $n'' \leftarrow$ end node of the $s[2 \ldots |s|]$-path from $\rho_{i+1}$ in $D_{i+1}$
            delete edge $(n, \sigma, n')$ from $D_i$ and insert edge $(n, \sigma, n'')$
Output final prefix DAG with roots $\rho_1, \ldots, \rho_{\ell-r+1}$:
    output $D \leftarrow D_1$
```

Figure 4: For a given self-set $S \subseteq \Sigma^\ell$ and number $r \in \{1, \ldots, \ell\}$, this procedure constructs a prefix DAG D with roots $\rho_1, \ldots, \rho_{l-r+1}$ such that $L(D, \rho_i) \cap \Sigma^{\ell-i+1} = \textsc{cont}(S, r)[i \ldots \ell]$ for every $i \in \{1, \ldots, \ell - r + 1\}$ in time $O(|S|\ell r|\Sigma|)$. Thus, in particular we have $L(D, \rho_1) \cap \Sigma^\ell = \textsc{cont}(S, r)$.

Construct prefix trees: For every $i \in \{1, \ldots, \ell - r + 1\}$, let T_i be the prefix tree with $L(T_i) \cap \Sigma^r = \Sigma^r \setminus S[i \ldots i + r - 1]$ from the proof of Theorem 3.1.

By definition we know that for every r-contiguous detector d and every $i \in \{1, \ldots, l - r + 1\}$, d contains a string at position i that occurs in T_i. However, there are still strings in T_i that do not occur in any r-contiguous detector at position i. Those are precisely the strings that have no (S, r)-avoiding left-completion or no (S, r)-avoiding right-completion. We will remove these strings in the upcoming two steps. For the correctness of this process, the following property of pairs (T_i, T_{i+1}) of adjacent trees is crucial:

Fact 4.3. *Let (T_i, T_{i+1}) be a pair of prefix trees on adjacent levels $i, i + 1$ from the proof of Theorem 3.1. For each $s \in T_i$ with $|s| \geq 2$, if there is no path from the root of T_{i+1} labeled with $s[2 \ldots |s|]$ then $s[2 \ldots |s|] \notin L(T_{i+1})$.*

Proof. Suppose the converse, i.e., $s[2 \ldots |s|] \in L(T_{i+1})$ and there is a proper nonempty prefix s' of $s[2 \ldots |s|]$ with $s' \in T_{i+1}$. The way that T_i was constructed ensures that for every $s \in T_i$, all of its proper prefixes occur in S at position i. This implies that s' occurs in S at position $i + 1$, which contradicts $s' \in T_{i+1}$.

Trim the trees in a right-to-left pass: We trim the trees $T_1, \ldots, T_{\ell-r+1}$ to obtain new trees $T^{\mathrm{R}}_1, \ldots, T^{\mathrm{R}}_{\ell-r+1}$ where every T^{R}_i contains exactly the strings from T_i that have (S, r)-avoiding right-completions. This holds directly for all strings from the rightmost level, so $T^{\mathrm{R}}_{\ell-r+1} = T_{\ell-r+1}$. We trim the other trees in a right-to-left pass from $i = \ell - r$ down to 1. Each time we initialize T^{R}_i to be the empty prefix tree. Then we consider every string $s \in T_i$ and insert it into T^{R}_i if $s[2 \ldots |s|]$ is a prefix of some $s' \in T^{\mathrm{R}}_{i+1}$. There are two potential reasons for a string $s \in T_i$ not to be contained in T^{R}_i: (1) It may be the case that no string from T_{i+1} starts with $s[2 \ldots |s|]$. Then, due to Fact 4.3 above, $s[2 \ldots |s|] \notin L(T_{i+1})$ which implies that all prefixes and suffixes of $s[2 \ldots |s|]$ with length $\leq r$ occur in S at position $i + 1$. Hence (s, i) has no

(S, r)-avoiding right-completion. (2) The second possibility is that there is a string that starts with $s[2 \ldots |s|]$ in T_{i+1}, but not in T^{R}_{i+1}. By induction, one can prove that this is due to the fact that $(s[2 \ldots |s|], i+1)$ has no (S, r)-avoiding right-completion and, therefore, also (s, i) has none. On the other hand, if $s \in T_i$ is also contained in T^{R}_i, there exists an (S, r)-avoiding right-completion of (s, i) consisting of overlapping strings from the trees $T^{\mathrm{R}}_i, \ldots, T^{\mathrm{R}}_{\ell-r+1}$. Thus, precisely those strings from T_i that have (S, r)-avoiding right-completions end up in T^{R}_i.

Trim the trees in a left-to-right pass: Next, we construct a set of trees $T^{\mathrm{L}}_1, \ldots, T^{\mathrm{L}}_{\ell-r+1}$ containing only the strings that have both left- and right-completions by an analogous left-to-right pass. Thus, $L(T^{\mathrm{L}}_i) \cap \Sigma^r = \textsc{cont}(S, r)[i \ldots i+r-1]$ holds.

Weave the trees together into a prefix DAG: Finally, we weave the trees together into a prefix DAG as follows: For the rightmost level $i = \ell - r + 1$, we set $D_{\ell-r+1} = T^{\mathrm{L}}_{\ell-r+1}$. This gives $L(T^{\mathrm{L}}_{\ell-r+1}) \cap \Sigma^r = \textsc{cont}(S, r)[\ell - r + 1 \ldots \ell]$. Now we prove the lemma by decreasing induction on i going from $i = \ell - r$ down to 1. For the induction step, assume we have a prefix DAG D_{i+1} with $L(D_{i+1}) \cap \Sigma^{\ell-i} = \textsc{cont}(S, r)[i+1 \ldots \ell]$. For $s \in T^{\mathrm{L}}_i$, let n' denote the corresponding leaf in T^{L}_i. Let n'' denote the end node on the path from the root of T^{L}_{i+1} with label s, which exists by induction assumption because $s[2 \ldots |s|]$ is a prefix of some $d \in \textsc{cont}(S, r)[i+1 \ldots \ell]$. Create a new edge from the parent n of n' to n'' and delete the leaf n' and the edge (n, σ, n'). After all leaves have been iterated through, let D_i be the resulting graph. Let $d \in \textsc{cont}(S, r)[i \ldots \ell]$. Then d starts with a prefix from T^{L}_i and, thus, $d[2 \ldots |d|] \in L(D_{i+1})$. Hence, $d \in L(D_i)$ by construction. Conversely, let $d \in L(D_i)$ with $|d| = \ell - i + 1$. Then d starts with a nonempty prefix that has both an (S, r)-avoiding right-completion and an (S, r)-avoiding left-completion. Furthermore, $d[2 \ldots |d|] \in L(D_{i+1})$. Hence $d \in \textsc{cont}(S, r)[i \ldots \ell]$. Now by setting $D = D_1$ we obtain a DAG with the properties claimed by the Lemma.

The runtime of the construction can be determined from the pseudocode given in Figure 4. As stated in Theorem 3.1, constructing the prefix trees in lines 1 and 2 takes time $O(|S|\ell r|\Sigma|)$. The inner loops in the right-to-left passes in lines 3–7 and 13–19 as well as in the left-to-right pass in lines 8–12 can be implemented by a simultaneous recursion through the trees on adjacent levels in time $O(|S|r|\Sigma|)$ per iteration. This yields a worst-case runtime of $O(|S|\ell r|\Sigma|)$ for each of the passes and, hence, of the overall algorithm. □

Proof (Theorem 4.1). Let D with roots $\rho_1, \ldots, \rho_{\ell-r+1}$ be the prefix DAG from Lemma 4.2. We transform D into an automaton $M = (Q, q_{\mathrm{i}}, Q_{\mathrm{a}}, \Sigma, \Delta)$ with $L(M) \cap \Sigma^\ell = \textsc{cont}(S, r)$: For every leaf n of D and $\sigma \in \Sigma$ we append a self-loop with label σ to n. Then Q and Δ are the set of nodes and set of labeled edges, respectively, Q_{a} contains all former leaves, and $q_{\mathrm{i}} = \rho_1$. Figure 5 shows an example of such an automaton. □

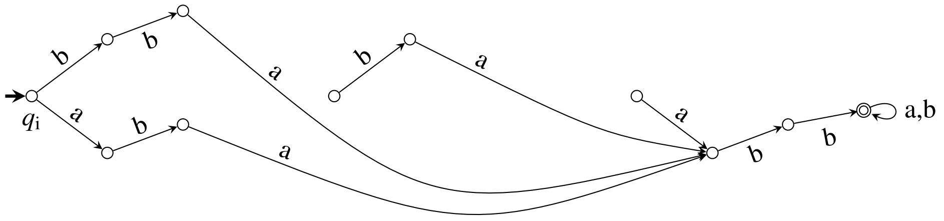


Figure 5: The constructed automaton M with $L(M) \cap \{a, b\}^5 = \textsc{cont}(S, 3)$ where S is the self-set from Figure 2.

In addition to describing the language $\textsc{cont}(S, r)$, the prefix DAG D can already be used to classify a string $m \in \Sigma^\ell$ in time $O(\ell r)$: Consider every position $i \in \{1, \ldots, \ell - r + 1\}$ and test whether $m[i \ldots i+r-1] \in L(D, \rho_i)$. If there exists a position where this is true, then m is "non-self" and "self", otherwise. At the end of this section we will speed up the classification to time $O(\ell)$. But first let us show how to use the prefix DAG D for counting the number of detectors.

Corollary 4.4. *There exists an algorithm that, given $S \subseteq \Sigma^\ell$ and $r \in \{1, \ldots, \ell\}$, outputs $|\textsc{cont}(S, r)|$ in time $O(|S|\ell r|\Sigma|)$.*

Proof. Our task is simply to count the number of strings of length ℓ in $L(D, \rho_1)$, where D is the prefix DAG constructed in Lemma 4.2. First, for each node $n \in D$, compute the number of different paths leading from ρ_1 to n. Denote this quantity by $P[n]$, and let $\delta(\rho_1, n)$ denote the distance between ρ_1 and n in D (note that by construction, all paths leading from ρ_1 to n in D have the same length). Then $|\textsc{cont}(S, r)| = \sum_{n \text{ is a leaf of } D} P[n] \cdot |\Sigma|^{\ell - \delta(\rho_1, n)}$. Since D is acyclic,

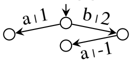
```
*Procedure* CONSTRUCT-CONTIGUOUS-NONSELF-MEALY-AUTOMATON($S$, $r$)
  $M \leftarrow$ Finite automaton from Theorem 4.1 with output 1 for all transitions
  $\rho_1, \dots, \rho_{\ell-r+1} \leftarrow$ root nodes of $M$'s graph
*Insert failure links with outputs in right-to-left pass*:
  **for** $i = \ell - r$ **down to** 1 **do**
      **for each** node $n$ reachable from $\rho_i$ but not from $\rho_{i+1}$ **do**
          **for each** $\sigma \in \Sigma$ where $n$ has no outgoing $\sigma$-edge **do**
              $p \leftarrow$ path from $\rho_i$ to $n$ ; $s \leftarrow$ string on $p$ ; $s' \leftarrow s\sigma$
              **if** there exists a path $p'$ for $s'[2 \dots |s'|]$ from $\rho_{i+1}$ **then**
                  $w \leftarrow$ sum of outputs on $p$ ; $w' \leftarrow$ sum of outputs on $p'$ ; $n' \leftarrow$ end node of $p'$
                  create a transition $(n, \sigma, n')$ with output $w' - w$
  **output** $M$
```

Figure 6: The procedure sketched in the proof of Theorem 4.6, which transforms the finite automaton M constructed by Theorem 4.1 into a Mealy automaton with $L(M,r) \cap \Sigma^\ell = \textsc{cont-nonself}(S,r)$. Note that the language $L(M,r)$, formalized in Definition 4.5, depends solely on the output of M, regardless its accepting states.

computing $P[n]$ can be done by a dynamic program that traverses D in breadth-first order from ρ_1. For the desired time bound note that the number of nodes and edges in D is bounded by $O(|S|\ell r|\Sigma|)$. □

Finally, let us discuss how to classify a single string in time $O(\ell)$. As in the previous section, this speedup can be achieved using failure links: We will augment the automaton constructed by Theorem 4.1 with edge outputs. The outputs will be numbers and their partial sums will equal the lengths of maximal partial matches to r-contiguous detectors. Formally, we use Mealy automata that output numbers and define a proper language based on these outputs.

Definition 4.5. A *Mealy automaton* is a tuple $M = (Q, q_\mathrm{i}, Q_\mathrm{a}, \Sigma, \Delta, \Omega, \omega)$ where $(Q, q_\mathrm{i}, Q_\mathrm{a}, \Sigma, \Delta)$ is a finite automaton, Ω is the *output alphabet*, and $\omega : \Delta \to \Omega$ is the *output function*. Let $m \in \Sigma^*$ and $t_1, \dots, t_{|m|} \in \Delta$ be the sequence of transitions made by M for input m, then the *output of M on input m* is the string $\omega(M,m) = \omega(t_1) \dots \omega(t_{|m|}) \in \Omega^*$. If Ω is a set of numbers, we define the *r-threshold language* $L(M,r)$ to be the set of strings $m \in \Sigma^*$ where there exists an $i \leq |m|$ with $\sum_{j=1}^{i} \omega(m)[j] \geq r$. Note that the definition of the threshold language does not refer to the accepting states of the automaton; it depends on the number outputs of the transitions.

Similar to finite automaton, a Mealy automaton can be represented by a graph where every edge label represents both the symbol that triggers the corresponding transition and the output of the transition. For example, for the Mealy automaton $M =$  we have $ba \in L(M,2)$ and $a \in L(M,1)$, but $a \notin L(M,2)$.

Theorem 4.6. *There exists an algorithm that, given any $S \subseteq \Sigma^\ell$ and $r \in \{1, \dots, \ell\}$, constructs a Mealy automaton M with output alphabet $\Omega = \{-r, \dots, r\}$ such that $L(M,r) \cap \Sigma^\ell = \textsc{cont-nonself}(S,r)$ in time $O(|S|\ell r|\Sigma|)$.*

Proof. Let M be the finite automaton constructed in the proof of Theorem 4.1 and let $\rho_1, \dots, \rho_{\ell-r+1}$ be the roots of its underlying graph. We turn M into a Mealy automaton with output alphabet $\Omega = \{-r, \dots, r\}$ such that $L(M,r) \cap \Sigma^\ell = \textsc{cont-nonself}(S,r)$ holds. We describe the main ideas of the construction and discuss its correctness. For a presentation of the detailed computation steps, we refer to the pseudocode in Figure 6. An example of the constructed automaton is shown in Figure 7.

We start by assigning to all existing transitions of M the output 1. Our aim is to transform M in a right-to-left pass that inductively ensures the following property: Let $m \in \Sigma^\ell$ and let $1 \leq i \leq j \leq \ell$. Let $k \geq 0$ denote the length of the longest suffix of $m[i \dots j]$ that is also a suffix of some $d' \in \textsc{cont}(S,r)[i \dots j]$. If $k \geq r - \ell + j$, then there exists a path from ρ_i for $m[i \dots j]$, and the sum of outputs on this path is equal to k. Otherwise there is no such path. Hence, if such a path exists and we have $k \geq r$, then $m \in \textsc{cont-nonself}(S,r)$; otherwise, k is the length of the longest partial match between $m[i \dots j]$ and some $d \in \textsc{cont}(S,r)[i \dots j]$ that can still be extended to length $\geq r$.

The property already holds for $i = \ell - r + 1$. For i decreasing from $\ell - r$ to 1, we iteratively transform the graph of M as follows: For every node n in M that is reachable from ρ_i, but not from ρ_{i+1}, consider all $\sigma \in \Sigma$ where n has no

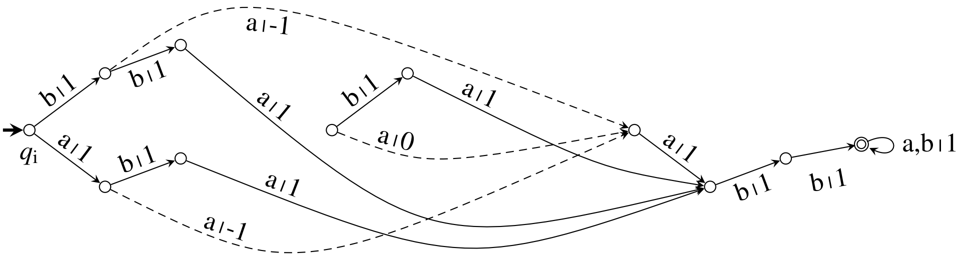


Figure 7: The Mealy automaton M with $L(M, 3) = \textsc{cont}(S, 3)$ where S is the self-set from Figure 2. The solid straight edges are the ones that remain from the initial prefix trees. The dashed lines are failure links, inserted to admit a linear time classification of a given string. Every edge is labeled with both the symbol that triggers the corresponding transition, and the number output of the transition.

outgoing edge labeled with σ. Let s be the string on the path p from ρ_i to n, w be the total weight on p, and $s' = s\sigma$. If there exists a path p' labeled with $s'[2 \ldots |s'|]$ from ρ_{i+1}, let w' denote the sum of weights on this path. Create an edge from n to the last node of p' and label it with $w' - w$. Now there is a path from ρ_i labeled with s' with weight w', satisfying the required property. The correctness of this procedure can be proved by induction, and we obtain a Mealy automaton whose r-threshold language has the desired property. Similarly as in Lemma 4.2, the described transformation can be implemented in time $O(|S|\ell r|\Sigma|)$ by simultaneous recursion from ρ_i and ρ_{i+1}. □

Assuming that we can add integers in unit time, we can compute the membership test for the r-threshold language $L(M, r)$ in time $O(\ell)$ and thus obtain a negative selection algorithm with time $O(|S|\ell r|\Sigma|)$ for the training phase and time $O(\ell)$ for classifying one string. However, it is possible to get rid of the unit cost assumption by using a finite automaton whose states store the values of the partial sums. For this construction, we would need to invest an additional runtime factor r in the training phase.

Corollary 4.7. *There exists an algorithm that, given any $S \in \Sigma^\ell$ and $r \in \{1, \ldots, \ell\}$, constructs a finite automaton M with $L(M) \cap \Sigma^\ell = \textsc{cont-nonself}(S, r)$ in time $O(|S|\ell r^2|\Sigma|)$.*

5. Conclusions

We have shown how to construct automata that simulate the classification results of negative selection algorithms with r-contiguous and r-chunk detectors. The constructions take time $O(|S|\ell r|\Sigma|)$ and enable subsequent classification of each string in linear time $O(\ell)$. Table 1 in the introduction compares the runtimes of previously published algorithms with those from the present paper. As a corollary, our result implies that the question if any r-contiguous detectors can be generated for a given self-set [14] can be answered in polynomial time. We leave it as an open problem whether the asymptotic time and space complexities of our constructions are optimal.

References

[1] J. O. Kephart, A biologically inspired immune system for computers, in: Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, MIT Press, 1994, pp. 130–139.

[2] S. Forrest, A. S. Perelson, L. Allen, R. Cherukuri, Self-nonself discrimination in a computer, in: Proceedings of the IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, 1994, pp. 202–212.

[3] C. Janeway, P. Travers, M. Walport, M. Shlomchick, Immunobiology, Garland Science, 2005.

[4] J. Balthrop, F. Esponda, S. Forrest, M. Glickman, Coverage and generalization in an artificial immune system, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002), 2002, pp. 3–10.

[5] S. Forrest, S. A. Hofmeyr, A. Somayaji, T. A. Longstaff, A sense of self for unix processes, in: Proceedings of the IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, USA, 1996, pp. 120–128.

[6] Z. Ji, D. Dasgupta, Revisiting negative selection algorithms, Evolutionary Computation 15 (2) (2007) 223–251. `doi:http://dx.doi.org/10.1162/evco.2007.15.2.223`.

[7] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Computing Surveys 41 (3) (2009) 1–58.

[8] C. M. Bishop, Novelty detection and neural network validation, IEE Proceedings on Vision and Image Signal Processing 141 (1994) 217–222.

[9] E. Parzen, On the estimation of a probability density function and mode, Annals of Mathematical Statistics 33 (1962) 1065–1076.

[10] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, R. C. Williamson, Estimating the support of a high-dimensional distribution, Neural Computation 13 (7) (2001) 1443–1471.

[11] T. Dunning, Statistical identification of language, Tech. rep., New Mexico State University (1994).

[12] W. Cavnar, J. M. Trenkle, N-gram-based text categorization, in: Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, 1994, pp. 161–175.

[13] M. Crochemore, F. Mignosi, A. Restivo, Automata and forbidden words, Information Processing Letters 67 (1998) 111–117.

[14] J. Timmis, A. Hone, T. Stibor, E. Clark, Theoretical advances in artificial immune systems, Theoretical Computer Science 403 (2008) 11–32.

[15] T. Stibor, On the appropriateness of negative selection for anomaly detection and network intrusion detection, Ph.D. thesis, Darmstadt University of Technology (2006).

[16] T. Stibor, P. Mohr, J. Timmis, C. Eckert, Is negative selection appropriate for anomaly detection?, in: Proceedings of the Genetic And Evolutionary Computation Conference (GECCO 2005), 2005, pp. 321–328.

[17] M. Elberfeld, J. Textor, Efficient algorithms for string-based negative selection, in: Proceedings of the 8th International Conference on Artificial Immune Systems (ICARIS 2009), Vol. 5666 of Lecture Notes in Computer Science, Springer, 2009, pp. 109–121.

[18] J. K. Percus, O. E. Percus, A. S. Perelson, Predicting the size of the T-cell receptor and antibody combining region from consideration of efficient self-nonself discrimination, Proceedings of the National Academy of Sciences of the United States of America 90 (5) (1993) 1691–1695. doi:{VL}-90.

[19] T. Stibor, Foundations of r-contiguous matching in negative selection for anomaly detection, Natural Computing 8 (2009) 613–641.

[20] P. D'haeseleer, S. Forrest, P. Helman, An immunological approach to change detection: Algorithms, analysis, and implications, in: Proceedings of the IEEE Symposium on Security and Privacy, IEEE Computer Society, 1996, pp. 110–119.

[21] P. D'haeseleer, An immunological approach to change detection: Theoretical results, in: Proceedings of the 9th IEEE Computer Security Foundations, IEEE Computer Society, 1996, pp. 18–26.

[22] S. T. Wierzchoń, Generating optimal repertoire of antibody strings in an artificial immune system, in: Intelligent Information Systems, Advances in Soft Computing, Physica-Verlag, 2000, pp. 119–133.

[23] M. Ayara, J. Timmis, R. de Lemos, L. N. de Castro, R. Duncan, Negative selection: How to generate detectors, in: J. Timmis, P. J. Bentley (Eds.), 1st International Conference on Artificial Immune Systems, Unversity of Kent at Canterbury Printing Unit, University of Kent at Canterbury, 2002, pp. 89–98.
URL http://www.cs.kent.ac.uk/pubs/2002/1504

[24] T. Stibor, K. M. Bayarou, C. Eckert, An investigation of r-chunk detector generation on higher alphabets, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), Vol. 3102 of Lecture Notes in Computer Science, Springer, 2004, pp. 299–307.

[25] T. Stibor, Phase transition and the computational complexity of generating r-contiguous detectors, in: Proceedings of the 6th International Conference on Artificial Immune Systems (ICARIS 2007), Vol. 4628 of Lecture Notes in Computer Science, Springer, 2007, pp. 142–155.

[26] U. Aickelin, Special issue on artificial immune systems – editorial, Evolutionary Intelligence 1 (2) (2008) 83–84.

[27] M. Liśkiewicz, J. Textor, Negative selection algorithms without generating detectors, in: Proceedings of Genetic and Evolutionary Computation Conference (GECCO'10), ACM, 2010, pp. 1047–1054.

[28] M. Crochemore, C. Hancart, T. Lecroq, Algorithms on Strings, 1st Edition, Cambridge University Press, 2007.

[29] D. E. Knuth, J. Morris, V. R. Pratt, Fast pattern matching in strings, SIAM Journal on Computing 6 (2) (1977) 323–350. doi:10.1137/0206024.