Negative Selection Algorithms Without Generating Detectors

Maciej Liśkiewicz
University of Lübeck
Institute for Theoretical Computer Science
Ratzeburger Allee 160
23538 Lübeck, Germany
liskiewi@tcs.uni-luebeck.de

Johannes Textor
University of Lübeck
Institute for Theoretical Computer Science
Ratzeburger Allee 160
23538 Lübeck, Germany
textor@tcs.uni-luebeck.de

ABSTRACT

Negative selection algorithms are immune-inspired classifiers that are trained on negative examples only. Classification is performed by generating *detectors* that match none of the negative examples, and these detectors are then matched against the elements to be classified. This can be a performance bottleneck: A large number of detectors may be required for acceptable sensitivity, or finding detectors that match none of the negative examples may be difficult. In this paper, we show how negative selection can be implemented without generating detectors explicitly, which for many detector types leads to polynomial time algorithms whereas the common approach to sample detectors randomly takes exponential time in the worst case.

In particular, we show that negative selection on strings with generating *all* detectors can be efficiently simulated without detectors if, and only if, an associated decision problem can be answered efficiently, regardless the detector type. We also show how to efficiently simulate the more general case in which only a limited number of detectors is generated. For many detector types, this non-exhaustive negative selection is more meaningful but it can be computationally more difficult, which we illustrate using Boolean monomials.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Pattern matching*; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Theory

Keywords

Artificial Immune Systems, Negative Selection, Consistent Learning

1. INTRODUCTION

Negative selection is the process by which the vertebrate immune system generates *T cells.* These cells can detect and destroy foreign substances, called *antigen* or *non-self.* The antigen-detecting receptors of T cells are generated randomly by recombination of DNA fragments. Newborn T cells migrate to the thymus, where they are confronted with bogus "antigens" that really are normal peptides from the organism, the *self.* Cells that detect such a "self-antigen" are bound to die, and only those that survive negative selection are allowed to leave the thymus and become detector cells.

This simple process contributes in several ways to the immune system's remarkable robustness. To name a few, detection is *distributed* – each T cell detects different antigen; and it is *diverse* – each individual of a species has its own unique set of T cells, making it difficult for pathogens to eradicate a species entirely. Naturally, such features are also highly desirable for computer security systems [2]. In the field of *artificial immune systems* (AIS), immunological processes are studied as a source of inspiration for engineered systems in general. Negative selection was one of the first processes considered, and substantial work was dedicated to it, comprehensively reviewed by Ji and Dasgupta [5].

Negative selection algorithms are quite verbatim abstractions of the actual process. Self and nonself are often represented as sets of strings with a fixed length L over some alphabet Σ. The detector cells are modeled as abstract entities that match subsets of the string universe Σ^L. For example, a detector d could itself be a string, which matches another string s if some distance $\delta(d, s)$ is sufficiently small. A generic outline of negative selection algorithms is shown in Figure 1: The input is a "self-set" S of strings representing normal behaviour. The algorithm first generates a set D of detectors that do not match any string in S. These detectors are to cover the "non-self" regions of Σ^L, and a string $m \in \Sigma^L$ is considered non-self if it is matched by a detector in D. Hence, negative selection is in principle a classification scheme.

In AIS literature, the two most common detector types for strings are the *r-chunk detectors* and the *r-contiguous detectors*, which we define later on. The computational complexity of generating such detectors for a given self-set was investigated in a series of papers [9, 10, 8], leading to the recent result by Elberfeld and Textor [1] that negative selection with these detectors is feasible in polynomial time, if we *avoid generating detectors explicitly.* In the present paper we generalize this idea to arbitrary detector types, which do not have to be associated with the immune system. Our main contributions are the following:

- We define negative selection as a *consistent learning algorithm* in the notational framework of algorithmic learning theory (Section 2).

- Since generating a large number of detectors is commonly assumed to improve the performance of negative selection algorithms, we first consider the case that *all* detectors are to be generated. This is also the case considered by Elberfeld and Textor [1], and we call it *exhaustive negative selection.* We show that the computational cost of this case is characterized by a special case of the *consistency problem* (Section 3), a decision problem. If this problem is efficiently solvable, we can use it to simulate exhaustive negative selection efficiently without generating detectors. However, we also give a natural detector type for which the problem is $\mathcal{NP}$-hard (Section 3.2).

- A more general approach is *non-exhaustive negative selection*, which generates only a limited number of detectors. We show how this can be simulated efficiently by *counting* detectors instead of generating them (Section 4). While we present the approach in a way that mimics the behaviour of negative selection with detector sampling, it can be easily derandomized. Again our approach works in particular for r-chunk and r-contiguous detectors, but we also give concrete examples for which it is unlikely to be computationally tractable.

- We show that there are detector types for which *exhaustive* negative selection is not meaningful, but *non-exhaustive* negative selection is actually a reasonable approach. In particular this is true for many detector types involving Boolean formulae. However, a derandomized simulation of non-exhaustive negative selection by counting detectors may be unlikely to be tractable computationally even when the *exhaustive* case can be simulated efficiently. We introduce *monotone monomial detectors* as a concrete example for this case.

2. BASIC TERMS AND DEFINITIONS

The terminology of negative selection literature closely reflects its source of inspiration, but most terms have equivalents in the algorithmic learning theory field. For example a "detector" would in learning theory be called "representation of a concept". In this paper, we use the terminology of algorithmic learning theory except that we adopt the terms *detector* and *detector type.* For reference, Table 1 shows equivalent terms from learning theory for the most important terms of negative selection, and Table 2 summarizes the symbols and notations used in the algorithms presented in this paper. In addition, we define the terminology formally in this section.

2.1 Strings

Let Σ be an alphabet and let $s \in \Sigma^L$ be a string consisting of L symbols from Σ. Then $|s| = L$ denotes the *length* of s and $s_i \in \Sigma$ is the i-th symbol of s. For two strings s and t, $\delta_H(s,t) = |\{i \mid s_i \neq t_i\}|$ denotes the Hamming distance.

Negative Selection	*Algorithmic Learning*
self, normal	negative example
non-self, anomalous	positive example
self-set	sample containing only negative examples
detector	representation of a concept
matching rule	membership function
detector type	concept class
detector generability	consistency problem

Table 1: Translation between terms used in the AIS literature about negative selection and the equivalent terms from algorithmic learning theory. This paper mainly uses terminology from algorithmic learning theory.

2.2 Concept classes and detectors

A *concept* c over Σ^L is any subset of Σ^L. A *concept class* or *detector type* $\mathcal{C} = \{c_1, c_2, \dots \mid c_i \subseteq \Sigma^L\}$ is any set of concepts with the following two properties. (1) $\mathcal{C}$ is *succinctly representable*: For every $c \in \mathcal{C}$ there exists a $d \in \Sigma^R$ that encodes c, where $R \in O(L)$. We call d a *detector* and write c_d for the concept represented by d. Furthermore, there is a polynomial-time algorithm that decides for every $d \in \Sigma^R$ whether d encodes a concept $c_d \in \mathcal{C}$. (2) $\mathcal{C}$ is *efficiently membership testable*: There is a polynomial-time algorithm $\chi_{\mathcal{C}}(x,d)$ with $\chi_{\mathcal{C}}(x,d) = 1$ if $x \in c_d$ and $\chi_{\mathcal{C}}(x,d) = 0$, otherwise[1]. We sometimes drop the distinction between a detector d and the corresponding concept c_d.

A *sample* $S \subseteq \Sigma^L \times \{+,-\}$ is a set of strings labeled with either "+" or "−". We take samples to be free of contradictions, i.e., there is no $x \in \Sigma^L$ with both $(x,+) \in S$ and $(x,-) \in S$. Furthermore, in the context of negative selection, we consider only samples that contain at most one positive example.

A concept c_d is said to be *consistent* with a sample S if $x \in c_d$ for all $(x,+) \in S$ and $x \notin c_d$ for all $(x,-) \in S$. For a sample S, $D_{\mathcal{C}}(S)$ denotes the set of all detectors d for which c_d is consistent with S. The *consistency problem* associated with a concept class $\mathcal{C}$ is a decision problem defined as follows: Given a sample S, reject the input if $D_{\mathcal{C}}(S)$ is empty, and accept otherwise. We say that a consistency problem is k^+*-restricted* if every possible input contains exactly k positive examples. The 0^+- and 1^+-restricted cases are most important in this paper.

The following two detector types are the most common ones in AIS [10]. They are inspired by the way that T cells in the immune system recognize antigen.

Definition 1. An *r-contiguous detector* $d \in \Sigma^L$ represents the concept

$$c_d = \{s \in \Sigma^L \mid \exists i : d_i \dots d_{i+r-1} = s_i \dots s_{i+r-1}\}$$

of all strings that share a substring of length r with d. An *r-chunk detector* $(d,i) \in \Sigma^r \times \{1, \dots, L-r+1\}$ represents

[1]Formally, concept classes $\mathcal{C}$ over Σ^L are understood here as *uniform families* of concept classes, for $L = 1, 2, 3, \dots$ that in principle should be indexed with parameter L. For sake of readability we skip the indices. However, we always consider the quantity L as variable. Thus, R in the requirement (1) above is considered as a function of L, and in the requirement (2) we assume that the membership test is done in time bounded by a polynomial with respect to L.

Notation	*Meaning*
$S \subseteq \Sigma^L \times \{-\}$	input sample set
$M \subseteq \Sigma^L$	input monitor set
$c \subseteq \Sigma^L$	concept over Σ^L
d and c_d	detector and the concept it represents
$\mathcal{C} = \{c_1, c_2, ...\}$	concept class or detector type
$D_\mathcal{C}(S)$	set of all detectors d for which c_d is consistent with S

Table 2: Symbols and notations used in the algorithms presented in this paper.

the concept

$$c_{d,i} = \{s \in \Sigma^L \mid s_i \dots s_{i+r-1} = d\}$$

of all strings in which d occurs at position i.

An obvious generalization of r-contiguous detectors is obtained by dropping the requirement for the matching string of length r between detector d and string s to be contiguous.

Definition 2. An *r-distributed detector* $d \in \Sigma^L$ represents the concept

$$c_d = \{s \in \Sigma^L \mid \exists\, i_1 < \dots < i_r : d_{i_1} \dots d_{i_r} = s_{i_1} \dots s_{i_r}\}$$

of all strings that share a distributed substring of length r with d. Equivalently, $s \in c_d$ if and only if $\delta_H(s, d) \leq L - r$.

In addition we define the following two detector types that are not immune-inspired, but based on Boolean monomials.

Definition 3. Let $\mathcal{L} = \{x_1, \bar{x}_1, \dots, x_L, \bar{x}_L\}$ be the set of literals. A *monomial* F over $\mathcal{L}$ is a conjunction $\lambda_1 \wedge \lambda_2 \wedge \dots \lambda_t$, such that each $\lambda_i \in \mathcal{L}$. Then F represents the concept

$$c_F = \{b \in \{0,1\}^L \mid F(b) = 1\}$$

of all assignments that satisfy the monomial F.

A *monotone monomial* F over $\mathcal{L}^+ = \{x_1, x_2, \dots, x_L\}$ is a monomial that contains only positive literals.

We assume that monomials are expressed in canonical form, i.e., there are no duplicate literals. This allows encoding every monomial F over $\mathcal{L} = \{x_1, \bar{x}_1, \dots, x_L, \bar{x}_L\}$ as a bit string of length $2L$, and every monotone monomial can be encoded as a bit string of length L.

2.3 Computational complexity preliminaries

We use in a standard way $\mathcal{N}P$-completeness as a yardstick for measuring the computational complexity of decision problems. For formal definitions of $\mathcal{N}P$-completeness and other standard notions of computational complexity, e.g. the complexity class $\mathcal{P}$ (deterministic polynomial time), see the textbook by Garey and Johnson [4].

For measuring computational complexity of counting problems in Section 4, which can be understood as functions $\Sigma^* \to \mathbb{N}$, we use $\#\mathcal{P}$-completeness as introduced by Valiant [11]. We define polynomial-time reductions between counting functions as follows. Let f and g be functions from Σ^* to $\mathbb{N}$. We say that f is *polynomial-time one-Turing reducible* to g if there is a pair of polynomial-time computable functions, $R_1 : \Sigma^* \to \Sigma^*$ and $R_2 : \Sigma^* \times \mathbb{N} \to \mathbb{N}$, such that for all x it holds that $f(x) = R_2(x, g(R_1(x)))$. Finally, we say that a counting problem Π is $\#\mathcal{P}$-hard if any problem in $\#\mathcal{P}$ is polynomial-time one-Turing reducible to Π. A problem is $\#\mathcal{P}$-complete if it is additionally contained in $\#\mathcal{P}$.

$\#\mathcal{P}$-completeness of a counting problem provides strong evidence about its intractability: Let Π be any $\#\mathcal{P}$-complete problem. For any problem Π' from the polynomial hierarchy, there exists a deterministic Turing machine M that solves Π' in polynomial time querying a Π-oracle at most once. In this sense, $\#\mathcal{P}$ is a "harder" complexity class than $\mathcal{NP}$, which is on the first level of the polynomial hierarchy.

```
NegativeSelection($S, M$).
Input: Sample $S \subseteq \Sigma^L \times \{-\}$, set $M \subseteq \Sigma^L$.
Output: For each $m \in M$, either $(m,+)$ or $(m,-)$.
1   $D \leftarrow$ some subset of $D_\mathcal{C}(S)$
2   for each $m \in M$ do
3       if there is a $d \in D$ with $m \in c_d$ then
4           output $(m,+)$
5       else
6           output $(m,-)$

ConsistentNegativeSelection($S, M$).
Input: Sample $S \subseteq \Sigma^L \times \{-\}$, set $M \subseteq \Sigma^L$.
Output: For each $m \in M$, either $(m,+)$ or $(m,-)$.
1   for each $m \in M$ do
2       if $|D_\mathcal{C}(S \cup \{(m,+)\})| > 0$ then
3           output $(m,+)$
4       else
5           output $(m,-)$
```

Figure 1: Above, general outline of a negative selection algorithm as it is usually defined [3, 10]. Below, exhaustive negative selection algorithm implemented without generating detectors explicitly. The output of this algorithm is equivalent to the specific version of the algorithm above in which *all* detectors are generated in line 1.

2.4 Negative selection algorithms

A negative selection algorithm is usually loosely defined as any algorithm that matches the generic scheme shown in Figure 1: Given a sample S of only negative examples and a "monitor set" M of strings to classify, generate a detector set $D \subseteq D_\mathcal{C}(S)$ consistent with S. Then for every string $m \in M$, label m positively if for some $d \in D$, c_d contains m.

Note that it is left to the implementation how to generate the detector set D. One very frequently used method is to sample detectors at random from $\mathcal{C}$, rejecting those that are inconsistent with S. This is repeated until the desired number of detectors is found. We start our analysis with the special case where *all* detectors are generated, which we call *exhaustive negative selection*. The more general case, i.e. *non-exhaustive negative selection*, will be considered in Section 4.

3. EXHAUSTIVE NEGATIVE SELECTION

In this section, we study the computational complexity of exhaustive negative selection. Even though it is in general infeasible to generate the entire detector set simply because it can have exponential size, the following proposition shows that we can simulate exhaustive negative selection without generating the detector set explicitly, regardless the type of detector used.

Proposition 1. *For every concept class $\mathcal{C}$, every sample $S \subseteq \Sigma^L \times \{-\}$, and every set $M \subseteq \Sigma^L$, the algorithms* NegativeSelection *with $D \leftarrow D_{\mathcal{C}}(S)$ in line 1 and* ConsistentNegativeSelection *(Figure 1) produce exactly the same output.*

Proof. The equivalence follows by definition of the consistency problem. □

Note that either algorithm simulates the other without any additional computational cost. Thus the 1^+-restricted consistency problem, which is a decision problem, characterizes the computational complexity of the entire negative selection algorithm as stated by the next corollary.

Corollary 1. *Exhaustive negative selection on a concept class $\mathcal{C}$ can be implemented in polynomial time if and only if the 1^+-restricted consistency problem for $\mathcal{C}$ can be solved in polynomial time.*

Hence an algorithm for the 1^+-restricted consistency problem can be used as a "black box" to implement exhaustive negative selection without generating detectors. In the rest of this section, we analyze the tractability of the 1^+-restricted consistency problem for important detectors types.

3.1 Tractable detector types

3.1.1 The r-chunk and r-contiguous detectors

As stated previously, most existing literature on negative selection on strings considers r-chunk detectors and r-contiguous detectors. The question if exhaustive negative selection can be efficiently implemented with these most common detector types was solved recently. In the framework of our analysis, the result can be stated as follows.

Theorem 1 (Elberfeld, Textor [1]). *The 1^+-restricted consistency problems for r-chunk and r-contiguous detectors are in $\mathcal{P}$.*

3.1.2 Boolean detectors

From the algorithmic learning theory perspective, both r-chunk and r-contiguous detectors are rather unusual concept classes. Perhaps the most important family of concept classes in the learning theory field are the *Boolean concept classes*, in which each concept is defined as the set of bitstrings that satisfy some formula ϕ. Depending on the type of formula used, this may give rise to $\mathcal{NP}$-hard consistency problems, e.g. for 3-term DNF [7]. However, as a consequence of the following proposition, most of these problems become trivial in the 1^+-restricted case.

Proposition 2. *Let $\mathcal{C}$ be any concept class that contains all sets $c \subset \Sigma^L$ with $|c| = 1$. Then the 0^+- and 1^+-restricted consistency problems for $\mathcal{C}$ are in $\mathcal{P}$.*

Proof. If S contains a positive example $(m, +)$, we know by definition that there is a concept containing exactly this example, and accept the input. If there is no positive example, we only need to check if there is at least one $x \in \Sigma^L$ that is not in S. Hence, our algorithm determines the number n of different negative examples in S and rejects the input if and only if $n = |\Sigma|^L$. The runtime of this algorithm is polynomial in the size of the input. □

As stated by the following corollary, this holds in particular for most canonical Boolean concept classes such as decision lists, decision trees, DNF or CNF formulae.

Corollary 2. *Let $\Sigma = \{0, 1\}$ and let $\mathcal{C}$ be any concept class over Σ^L subsumed by the monomials. Then the 1^+-restricted consistency problem for $\mathcal{C}$ is in $\mathcal{P}$.*

Proof. For every string $s \in \{0, 1\}^L$, we can construct a monomial that evaluates to true only on s: Simply set $\lambda_i = x_i$ if $s_i = 1$ and $\lambda_i = \bar{x}_i$, otherwise. □

This means that for Boolean detectors, exhaustive negative selection is useless as a classification algorithm: Every monitor string $m \notin S$ is positively labeled, since there is always a detector d with c_d containing only m. It is not surprising that the associated consistency problem is trivial. However, we will see later on that using Boolean detectors is reasonable (but not necessarily computationally tractable) for *non-exhaustive* negative selection.

The property of Proposition 2 does not apply to monotone monomials, which are less expressive than general monomials. Still, monotone monomials do admit polynomial time exhaustive negative selection.

Proposition 3. *The 0^+- and 1^+-restricted consistency problems for monotone monomials are in $\mathcal{P}$.*

Proof. For the 0^+-restricted consistency problem, let F be the monomial $x_1 x_2 \dots x_L$ containing all literals. F evaluates to true only on the string 1^L. Hence, if $(1^L, -) \notin S$, accept the input. Otherwise reject the input, since $(1^L, -) \in S$ and there is no monotone monomial F such that $F(1^L) = 0$. This test can be performed in polynomial time.

For the 1^+-restricted consistency problem, let m be the unique positive example where $(m, +) \in S$. If $m = 0^L$ then reject the input since there exists no monotone monomial satisfied by 0^L. Next, assume string m has at least one bit 1. Any monotone monomial satisfied by m contains only literals x_i where $m_i = 1$. Let $I = \{i_1, \dots, i_k\}$, $0 < k \leq L$, be the set of all these indices, i.e., $i \in I$ if and only if $m_i = 1$. Now consider the reduced sample S' obtained from S by deleting from each negative example string $s \in S$ the letters at positions $i \notin I$, i.e., the sample

$$S' = \{(s_{i_1} \dots s_{i_k}, -) \mid (s, -) \in S\}$$

and solve the 0^+-restricted consistency problem for S'. Computing S' from S can be done in polynomial time. □

3.2 Intractability of r-distributed detectors

To our knowledge, no explicit examples of detector types for which the 1^+-restricted consistency problem is $\mathcal{NP}$-hard have been given in the literature, even though this was suspected to be the case for r-contiguous detectors [10]. Here we give an example that arises naturally through a generalization of r-contiguous detectors.

Theorem 2. *The 0^+-restricted and 1^+-restricted consistency problems for r-distributed detectors are $\mathcal{NP}$-complete.*

Proof. The 0^+-restricted consistency problem for r-distributed detectors can be equivalently stated as follows: For a set S of strings, find a string d such that $\delta_H(d, s) > L - r$ for all $s \in S$. This is the well-known *farthest string problem* [6],

```
SamplingNegativeSelection($S, M, n$).
Input: Sample $S \subseteq \Sigma^L \times \{-\}$, set $M \subseteq \Sigma^L$, integer $n$.
Output: For each $m \in M$, either $(m,+)$ or $(m,-)$.
D ← ∅ and i ← 0
while $i < n$ do
    pick $d \in \mathcal{C}$ uniformly at random
    if $c_d$ is consistent with $S$ then
        $D \leftarrow D \cup \{d\}$ and $i \leftarrow i+1$
for each $m \in M$ do
    if there is any $d \in D$ with $m \in c_d$ then
        output $(m,+)$
    else
        output $(m,-)$
```

```
RandomNegativeSelection($S, M, n$).
Input: Sample $S \subseteq \Sigma^L \times \{-\}$, set $M \subseteq \Sigma^L$, integer $n$.
Output: 'undefined' or for each $m \in M$, either
    $(m,+)$ or $(m,-)$.
if $|D_{\mathcal{C}}(S)| = 0$ then
    output 'undefined'
else
    for each $m \in M$ do
        compute $p = (1 - \delta_{\mathcal{C}}(S,m))^n$
        with probability $1 - p$ do
            output $(m,+)$
        else do
            output $(m,-)$
```

Figure 2: Above, negative selection algorithm that generates detectors by random sampling, which is common in practice [5]. Below, randomized implementation of sampling negative selection without generating detectors explicitly.

which is $\mathcal{NP}$-complete. For the 1^+-restricted consistency problem, we have the additional constraint $\delta_H(d,m) \le r$, where m is the unique positive example. It is easily seen that despite the additional constraint the problem remains $\mathcal{NP}$-hard. □

4. NON-EXHAUSTIVE NEGATIVE SELECTION

Most existing implementations of negative selection are not exhaustive, but generate only a certain number of detectors by random sampling (Figure 2). The number n of detectors to be generated is part of the input and controls the algorithm's sensitivity: The larger the value of n, the more probably each string $m \in M$ is labeled positively. Exhaustive negative selection maximizes this sensitivity. However, a too large sensitivity might result in a classifier that *overfits*, and thus it can be desirable to work with incomplete detector sets. In this section, we ask under which circumstances sampling negative selection can be simulated *without generating detectors* – i.e., turned into a randomized algorithm that is guaranteed to terminate and does so in polynomial worst-case time.

A generic outline of sampling negative selection is shown in Figure 2. Note that we assume that the generated detectors are not checked for duplicates. All results in this section still hold if we do, but the analysis becomes more technical at some points highlighted later on. Also we sample the detectors *uniformly* from $\mathcal{C}$, which is straightforward for the concept classes discussed in this paper (and all those from the AIS literature) but may be highly nontrivial for other concept classes such as arbitrary boolean formulae.

Definition 4. Let $\mathcal{C}$ be a concept class, $S \subseteq \Sigma^L \times \{-\}$ a sample consisting of negative examples with $|D_{\mathcal{C}}(S)| > 0$, and m a string. Then the *detector sampling distance* $\delta_{\mathcal{C}}(S,m)$ is defined by

$$\delta_{\mathcal{C}}(S,m) = \frac{|D_{\mathcal{C}}(S \cup \{(m,+)\})|}{|D_{\mathcal{C}}(S)|} .$$

We interpret $\delta_{\mathcal{C}}(S,m)$ as a distance because the more structure is shared between m and S, the less likely a concept chosen at random from "outside" of S contains m. In particular, $\delta_{\mathcal{C}}(S,m) = 0$ if $m \in S$.

Using the detector sampling distance we are able to express the probabilities that algorithm SamplingNegativeSelection labels every string m positively or negatively if the algorithm terminates.

Proposition 4. *For every concept class $\mathcal{C}$ over Σ^L, every sample $S \subseteq \Sigma^L \times \{-\}$, and every $M \subseteq \Sigma^L$, algorithm* SamplingNegativeSelection *on input (S,M,n) does not terminate[2] if $|D_{\mathcal{C}}(S)| = 0$. Otherwise, the algorithm terminates and for each $m \in M$ it classifies m negatively with probability $p = (1 - \delta_{\mathcal{C}}(S,m))^n$, and positively with probability $1 - p$.*

Proof. If $|D_{\mathcal{C}}(S)| = 0$, algorithm SamplingNegativeSelection obviously does not terminate. Otherwise it terminates and classifies every $m \in M$ either positively or negatively. Assume algorithm SamplingNegativeSelection terminates on input (S,M,n) and let k denote the number of detectors from D after line 5 of the algorithm that are consistent with $S \cup \{(m,+)\}$. Then k is binomially distributed[3]. The probability that none of the generated detectors contains the string m is $\Pr[\,k = 0] = (1 - \delta_{\mathcal{C}}(S,m))^n$. The probability that one of the generated detectors contains m is $\Pr[\,k > 0] = 1 - \Pr[\,k = 0]$, which completes the proof. □

Corollary 3. *For every concept class $\mathcal{C}$ over Σ^L, every sample $S \subseteq \Sigma^L \times \{-\}$, and every $M \subseteq \Sigma^L$, algorithm* SamplingNegativeSelection *does not terminate if and only if algorithm* RandomNegativeSelection *outputs "undefined". Moreover, if algorithm* SamplingNegativeSelection *terminates, both algorithms label each m positively with exactly the same probabilities.*

Note that algorithm RandomNegativeSelection does not generate any detectors explicitly, but computing $\delta_{\mathcal{C}}$ involves *counting* the detector sets $D_{\mathcal{C}}(S)$ and $D_{\mathcal{C}}(S \cup \{(m,+)\})$ for each $m \in M$. Hence, the complexity of this procedure is characterized by a *counting problem.* This leads to the counterpart of Corollary 1 for non-exhaustive negative selection.

Corollary 4. SamplingNegativeSelection *on a concept class $\mathcal{C}$ can be implemented in polynomial time if (but not necessarily only if) the 0^+-restricted consistency problem for $\mathcal{C}$ and the concept sampling distance $\delta_{\mathcal{C}}$ are computable in polynomial time.*

[2]If we check for duplicate detectors, sampling negative selection does in addition not terminate for $|D_{\mathcal{C}}(S)| < n$.

[3]This becomes a hypergeometric distribution if we check for duplicate detectors

Note that we can *derandomize* the algorithm RANDOM-NEGATIVESELECTION by directly outputting each $\delta_{\mathcal{C}}(S, m)$ in line 5. This would lead to a nice deterministic generalization of negative selection in which each string is not labeled with either $+$ or $-$, but with a value from the interval $[0, 1]$ that indicates how "far away" m is from the set S.

In the rest of this section, we investigate the complexity of computing $\delta_{\mathcal{C}}$ for some important detector types.

4.1 Tractable detector types

4.1.1 The r-chunk detectors

For r-chunk detectors we can easily compute $\delta_{\mathcal{C}}(S, m)$ as stated by the following proposition.

THEOREM 3. *Let $\mathcal{C}$ be the concept class of r-chunk detectors. Then for all samples $S \subseteq \Sigma^L \times \{-\}$ and all strings m,*

$$\delta_{\mathcal{C}}(S, m) = \frac{\sum_i \min\{1, \min_{s \in S}\{\delta_H(s_{i \dots i+r-1}, m_{i \dots i+r-1})\}\}}{\sum_i |\Sigma|^r - |\{s_i \dots s_{i+r} \mid s \in S\}|}$$

PROOF. The expression is correct because a string m can match at most one r-chunk detector per index $i \in \{1, \dots, L - r + 1\}$ (numerator), and the number of r-chunk detectors consistent to S per index i is equal to the number of strings of length r that do not occur in any $s \in S$ (denominator). $\square$

Note that we compute $\delta_{\mathcal{C}}(S, m)$ merely by inspecting the input dataset S, and this does not require any detector to be generated and stored explicitly.

4.1.2 The r-contiguous detectors

For r-contiguous detectors it is also possible to compute the concept sampling distance efficiently, although this requires a little more effort than for r-chunk detectors.

THEOREM 4. *Let $\mathcal{C}$ be the concept class of r-contiguous detectors on the alphabet $\Sigma = \{0, 1\}$. Then the concept sampling distance $\delta_{\mathcal{C}}(S, m)$ can be computed in polynomial time.*

PROOF. For our proof we build upon the pattern graph construction by Elberfeld and Textor [1]. This is not generalizable to higher alphabets, but for reasons of clarity and conciseness we refrain from taking a more generic approach. The concepts in this proof are illustrated in Figure 3. In a pattern graph G, every path from the leftmost to the rightmost level describes a set of detectors, which is obtained by merging the patterns on the path. For example, merging the patterns $(00\diamond, 1), (0\diamond\diamond, 2), (100, 3)$ from the corresponding path in G in Figure 3 yields the pattern 00100, which describes only one detector. This merging process makes counting the number of detectors difficult, thus we need to transform the graph slightly for this purpose.

Let G be an r-contiguous pattern graph with $L - r + 1$ levels. We denote by $G[i]$ the set of vertices on level i of this graph. Then the *r-contiguous counting graph* G' is constructed from G as follows: (1) Copy all vertices and edges from G to G'. (2) For every vertex $v \in G$ labeled with a pattern $\pi = \pi_1 \dots \pi_r$, label the corresponding vertex $v' \in G'$ with π_1. (3) For every vertex $v \in G[L - r + 1]$ do the following: Denote the corresponding vertex in G' by v'_1. For $i \in \{2, \dots, r\}$, create a vertex v'_i labeled with π_i and an edge (v'_{i-1}, v'_i).

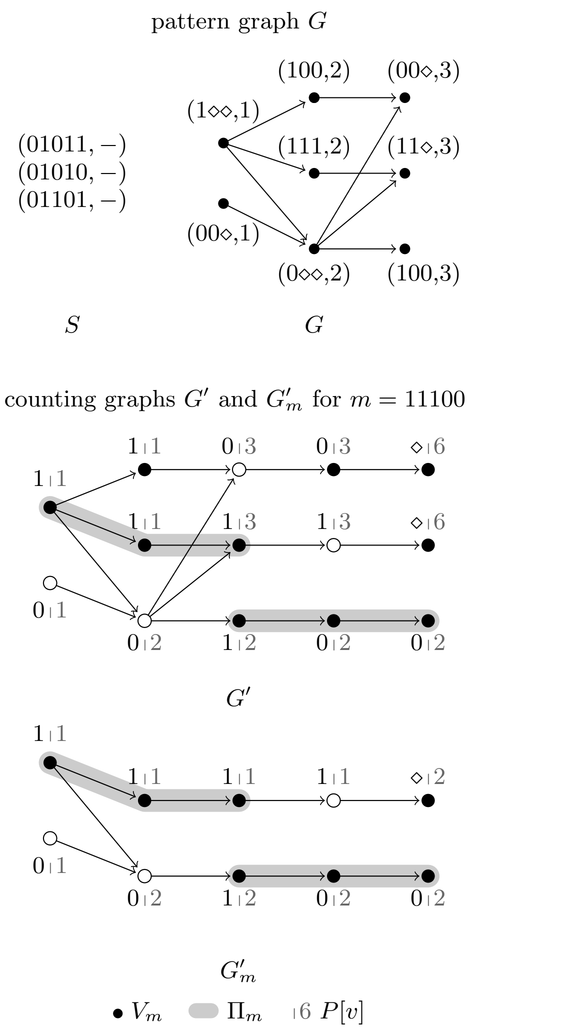

Figure 3: Computing the detector sampling distance for r-contiguous detectors as per Theorem 4. For the shown sample S and $r = 3$, we have $|D_{\mathcal{C}}(S)| = 14$ by summation of $P[v]$ on level 5 of G', and then $|D_{\mathcal{C}}(S \cup \{(m, +)\})| = 4$ for G'_m. Hence $\delta_{\mathcal{C}}(S, m) = 4/14$.

The resulting r-contiguous counting graph G' has L levels. For every path p from the leftmost to the rightmost level in G', concatenating all vertex labels on p (left part of vertex annotations in Figure 3) yields the merged pattern from the corresponding path in G. We obtain the theorem through the following two claims.

Claim 1. Given an r-contiguous counting graph G', we can compute the number of detectors described by G' in polynomial time.

Proof. We compute the number of detectors by dynamic programming. (1) For every vertex on level 1 of G', set $P[v] = 2$ if v is labeled with $\diamond$ and 1, otherwise. (2) For each $i \in \{2, \dots, L\}$ consider the set of vertices v on level i and set

$$P[v] = \sum_{u \in \mathrm{pre}(v)} P[u] \times \begin{cases} 2 & v \text{ is labeled with } \diamond \\ 1 & \text{otherwise} \end{cases}$$

where $\mathrm{pre}(v)$ is the set of vertices on level $i - 1$ that are linked to v. Computing $P[v]$ for all $v \in G'$ in this manner takes polynomial time. The number of detectors encoded by G' is then given by the expression $\sum_{v \in G'[L]} P[v]$.

Claim 2. For every r-contiguous counting graph G' and every string $m \in \Sigma^L$, there is a subgraph $G'_m \subseteq G'$ describing only the detectors encoded by G' that also match m. Furthermore, G'_m can be computed from G' in polynomial time.

Proof. The following algorithm constructs G'_m from G'. (1) For $i \in \{1, \dots, L\}$, let $V_m[i] \subseteq G'[i]$ denote the set of vertices on level i in G' labeled with either $\diamond$ or m_i, and let $V_m = \bigcup_i V_m[i]$. (2) Let Π_m be the set of all paths in G' consisting of exactly r vertices from V_m. These paths need not be disjoint. (3) Delete all edges (u, v) from G' for which none of the following holds: (i) (u, v) is on some path $p \in \Pi_m$; (ii) The leftmost vertex of some path $p \in \Pi_m$ is either v or reachable from v; (iii) The vertex u is either the rightmost vertex of some path $p \in \Pi_m$ or it is reachable from such a vertex.

The resulting graph G'_m contains every path $p \in \Pi_m$, and every path from the leftmost to the rightmost level contains at least one subpath from Π_m. Hence it describes all detectors that match m.

Because $\delta_{\mathcal{C}}(S, m) = \left(\sum_{v \in G'_m[L]} P[v]\right) / \left(\sum_{v \in G'[L]} P[v]\right)$, the theorem follows. $\square$

4.2 Intractable detector types

4.2.1 The r-distributed detectors

As an example concept class for which non-exhaustive negative selection is unlikely to be efficiently implementable using our approach, we use again the r-distributed detectors (Definition 2). For this concept class, we have seen that the 0^+-restricted consistency decision problem is $\mathcal{NP}$-complete (Theorem 2). Hence, one cannot expect that the consistency test in line 1 of the algorithm RandomNegativeSelection (Figure 2) can be implemented efficiently. But we can still be interested in computing the probabilities that sampling negative selection algorithm classifies m positively if the algorithm terminates. The following theorem gives evidence that this task can be even harder than solving the associated decision problem.

Theorem 5. *Let $\mathcal{C}$ be the concept class of r-distributed detectors (Definition 2). Then computing the detector sampling distance $\delta_{\mathcal{C}}$ is $\#\mathcal{P}$-complete.*

Proof (Sketch). First note that our theorem adopts a somewhat broader interpretation of $\#\mathcal{P}$-completeness, since according to our definition in section 2.3 only functions from Σ^* to the natural numbers can be $\#\mathcal{P}$-complete. However, for any sample S and string m, computing $|D_{\mathcal{C}}(S)|$ and $|D_{\mathcal{C}}(S \cup \{(m, +)\})|$ is both in $\#\mathcal{P}$. Hence we can compute $\delta_{\mathcal{C}}$ by making two queries to a $\#\mathcal{P}$-oracle.

We will prove the $\#\mathcal{P}$-hardness of computing $\delta_{\mathcal{C}}$ showing that counting the number of satisfying assignments for a 2-CNF formula, which is $\#\mathcal{P}$-complete [11], is polynomial-time one-Turing reducible to computing $\delta_{\mathcal{C}}$. Due to space limitations we give only a sketch for this reduction and skip the analysis. The reduction uses however some key ideas from the one given by Lanctot et al. [6] to prove $\mathcal{NP}$-hardness of the farthest string problem.

Assume F is a 2-CNF formula. Let $x_1, x_2, \dots, x_\ell$ denote the variables in F and let k be the number of clauses of F. We perform the reduction in two steps. In the first one we construct a 3-CNF formula F' over variables $x_0, x_1, x_2, \dots, x_\ell$ (hence we add one new variable x_0) and in the second step we construct for the formula F' a sample S and a string m. The crucial property of this construction is that

$$\delta_{\mathcal{C}}(S, m) = \frac{\#\mathrm{SAT}(F'|_{x_0=0})}{\#\mathrm{SAT}(F')} = \frac{\#\mathrm{SAT}(F)}{\#\mathrm{SAT}(F) + 2^\ell},$$

where $F'|_{x_0=0}$ denotes the formula in which x_0 is substituted by the constant 0 and $\#\mathrm{SAT}(F)$ is the number of satisfying assignments for the formula F. Thus, one computes the number of satisfying assignments for the formula F by

$$\#\mathrm{SAT}(F) = \frac{\delta_{\mathcal{C}}(S, m) \cdot 2^\ell}{1 - \delta_{\mathcal{C}}(S, m)}.$$

We transform F to F' by adding to each clause of F the new variable x_0. Thus e.g. for $F = (x_1 \vee \bar{x_2}) \wedge (\bar{x_1} \vee \bar{x_3})$ we get $F' = (x_0 \vee x_1 \vee \bar{x_2}) \wedge (x_0 \vee \bar{x_1} \vee \bar{x_3})$. The reduction of F' to S and m is more involved. The set S encodes clauses of F' and some constraints to enforce that every detector d that matches none of the strings in S has the form $(01 \mid 10)^*$. This way, d can be interpreted as an assignment for the variables in F. The string for m enforces that the variable x_0 is set to 0.

More explicitly, the set S contains strings of length $2 \cdot (\ell + 2)$. It contains (1) words $(11)^{\ell+2}$ and $(00)^{\ell+2}$, (2) words $(11)^i 00 (11)^{\ell+1-i}$ for $i = 0, \dots, \ell+1$, (3) the word $10(00)^{\ell+2}$ and (4) for every clause $(x_0 \vee \lambda \vee \lambda')$, if λ is a literal over the variable x_i and λ', over x_j, with $i < j$, we add to S the word $1010(00)^{i-1} u (00)^{j-1-i} u' (00)^{\ell+2-j}$ such that $u = 10$ if $\lambda = x_i$ and $u = 01$, otherwise. The word u' encodes the literal λ' analogously. Finally we define the word m as $0010(00)^\ell$ and specify $r = \ell + 2$. $\square$

4.2.2 Monotone monomial detectors

For the r-distributed detectors, we have shown evidence that neither exhaustive negative selection nor derandomized non-exhaustive negative selection are computationally tractable. Finally, we show an example that illustrates the less intuitive situation where *exhaustive* negative selection is *tractable*, but derandomized *non-exhaustive* negative selection is *intractable* unless $\#\mathcal{P} = \mathcal{P}$.

We have seen in Proposition 3 that monotone monomial detectors admit polynomial-time exhaustive negative selection. Hence, we can compute the consistency test in line 1 of the algorithm RandomNegativeSelection (Figure 2) in polynomial time. However, the detector sampling distance is unlikely to be efficiently computable, as shown by the following theorem.

Theorem 6. *Let $\mathcal{C}$ be the concept class of the monotone monomials (Definition 3). Then computing the detector sampling distance $\delta_{\mathcal{C}}$ is $\#\mathcal{P}$-complete.*

Proof. Similarly to the previous theorem, the proof proceeds by showing that counting the number of satisfying assignments of *monotone* 2-CNF formulae is polynomial time one-Turing reducible to computing $\delta_{\mathcal{C}}$. Analogously to monotone monomials, monotone 2-CNF formulae contain only positive literals. It was shown by Valiant that the associated counting problem is $\#\mathcal{P}$-complete even for this very restricted formula class [11].

Let Φ be a monotone 2-CNF formula over the literals $\mathcal{L} = \{x_1, \dots, x_L\}$ containing k clauses. We can assume that every literal is contained at least once in Φ (otherwise we can rename literals accordingly and use a smaller L). For example,

$$\Phi = c_1 \wedge c_2 \wedge c_3 = (x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (x_3 \vee x_3)$$

is such a formula and it has three satisfying assignments, namely $a_1 = 011$, $a_2 = 101$, and $a_3 = 111$ ($L = k = 3$). We construct a negative example set S for Φ by converting every clause c in Φ to a string as follows: For each $i \in \{1 \dots L\}$, set $s_i = 0$ if $x_i \in c$ and $s_i = 1$, otherwise. Finally, set $s_{L+1} = 0$. For the above example,

$$S = \{(0100, -), (0010, -), (1100, -)\} \quad .$$

Now, every monotone monomial F consistent with S falls in one of the following two categories:

1. $x_{L+1} \in F$. There are 2^L monomials with this property.
2. $x_{L+1} \notin F$. Because F is consistent with S, it corresponds to a satisfying assignment a of Φ as follows: Set $a_i = 1$ if $x_i \in F$ and $a_i = 0$, otherwise.

On the other hand, for every satisfying assignment a for Φ there is exactly one monomial consistent with S with $x_{L+1} \notin F$ (e.g. for S given above, $F_1 = x_2x_3$ corresponds to $a_1 = 011$, $F_2 = x_1x_3$ corresponds to $a_2 = 101$, and $F_3 = x_1x_2x_3$ corresponds to $a_3 = 111$). Hence, $|D_\mathcal{C}(S)| = \#\mathrm{SAT}(\Phi) + 2^L$. Now we set

$$m = \underbrace{00\dots0}_{L}1 \quad .$$

It is easy to see that the only monotone monomial satisfying m is x_{L+1}. This monomial is also consistent with S. Hence, $|D_\mathcal{C}(S \cup \{(m, +)\})| = 1$. Hence, if we have access to an oracle for $\delta_\mathcal{C}(S, m)$, we can infer $\#\mathrm{SAT}(\Phi)$ through the equation

$$\delta_\mathcal{C}(S, m) = \frac{1}{\#\mathrm{SAT}(\Phi) + 2^L} \quad .$$

Since constructing S and m from Φ as well as solving the above equation takes polynomial time, this completes the proof. $\square$

Hence, this result is a consequence of the high difficulty of exactly determining the number of solutions to Boolean formulae even for the very restricted class of monotone 2-CNF formulae. Note however, that this example does not immediately generalize to other types of Boolean detectors – e.g. computational hardness of computing the detector sampling distance for *arbitrary* monomials cannot be established using the same reduction.

5. CONCLUSIONS AND FUTURE WORK

We have formalized string-based negative selection with arbitrary detector types as a consistent learning algorithm in the notational framework of algorithmic learning theory. From our results it appears that the complexity landscape of both exhaustive and non-exhaustive negative selection algorithms is equally rich as for consistent learning in general.

There are detector types for which an efficient implementation of at least exhaustive negative selection is unlikely to exist. However, for the most commonly used detector types in artificial immune systems, both exhaustive and non-exhaustive negative selection are implementable in polynomial time using the techniques we presented to avoid generating detectors explicitly.

We would like to emphasize that our work generalizes negative selection on strings beyond the common detector types that have their roots in theoretical immunology. The practical use cases for r-chunk and r-contiguous detectors are limited because both assume a semantic correlation between adjacent positions of the input strings, which need not be the case in real-world datasets. Detectors based on boolean formulae do not present this problem. Future work should address the question if better classification performance can be obtained on real-world datasets using non-exhaustive negative selection with such detector types. If that is the case, it should be studied whether approximation schemes or fixed parameter algorithms can be applied to overcome the potential computational complexity issues illustrated in Section 4.2.

6. REFERENCES

[1] M. Elberfeld and J. Textor. Efficient algorithms for string-based negative selection. In *Proc. of ICARIS 2009*, volume 5666 of *LNCS*, pages 109–121. Springer, 2009.

[2] S. Forrest, S. A. Hofmeyr, and A. Somayaji. Computer immunology. *Communications of the ACM*, 40:88–96, 1997.

[3] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, pages 202–212. IEEE Press, 1994.

[4] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. Freeman and Company, 1979.

[5] Z. Ji and D. Dasgupta. Revisiting negative selection algorithms. *Evolutionary Computation*, 15(2):223–251, 2007.

[6] J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Information and Computation*, 185:41–55, 2003.

[7] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35:965–984, 1988.

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

[9] T. Stibor, J. Timmis, and C. Eckert. The link between r-contiguous detectors and k-cnf satisfiability. In *Proc. of GECCO 2006*, pages 491–498. ACM, 2006.

[10] J. Timmis, A. Hone, T. Stibor, and E. Clark. Theoretical advances in artificial immune systems. *Theoretical Computer Science*, 403:11–32, 2008.

[11] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8:410–421, 1979.