Efficient Algorithms for String-Based Negative Selection

Michael Elberfeld and Johannes Textor

Institut für Theoretische Informatik
Universität zu Lübeck, 23538 Lübeck, Germany
{elberfeld,textor}@tcs.uni-luebeck.de

Abstract. String-based negative selection is an immune-inspired classification scheme: Given a self-set S of strings, generate a set D of detectors that do not match any element of S. Then, use these detectors to partition a monitor set M into self and non-self elements. Implementations of this scheme are often impractical because they need exponential time in the size of S to construct D. Here, we consider r-chunk and r-contiguous detectors, two common implementations that suffer from this problem, and show that compressed representations of D are constructible in polynomial time for any given S and r. Since these representations can themselves be used to classify the elements in M, the worst-case running time of r-chunk and r-contiguous detector based negative selection is reduced from exponential to polynomial.

1 Introduction

The immune system protects us from many dangerous pathogens it has never seen, and it does that by essentially playing dice: T cells are generated randomly and in large numbers, in the hope that every pathogen that infects the host is detected by at least some of these cells. However, the host must ensure that no cells are generated that would turn against itself – many severe diseases are caused by such autoimmune reactions. Hence, newborn T cells undergo the process of *negative selection.* In a special organ, the thymus, they are shown *self* proteins, which belong to the host. If a T cell detects any self protein, it is destroyed.

When Forrest et al. [1, 2] analyzed the immune system as a source of inspiration for computer security, they found that the problem faced by the immune system is similar to one that today's computer systems face: It is difficult to defend a system against a previously unknown danger, such as an exploit of a new security hole. The only reliable knowledge we have is the normal behavior of the system – the equivalent of self. The idea of the negative selection classification scheme is to mimic the T cells in the immune system: Generate a set of detectors that do not match anything in self, then use these detectors to monitor the system for unusual behavior. A more precise definition of the negative selection algorithm is shown in Figure 1.

Algorithm NEGATIVE-SELECTION.
Input: A self-set $S \subseteq \mathcal{U}$, a monitoring set $M \subseteq \mathcal{U}$.
Output: For each element $m \in M$, either self or non-self.
1 $D \leftarrow$ Set of detectors that do not match any $s \in S$.
2 **for each** $m \in M$ **do**
3 **if** m matches any detector $d \in D$ **then**
4 **output** "m is non-self"
5 **else**
6 **output** "m is self"

Fig. 1. The generic negative selection algorithm.

The generic scheme of negative selection is independent of the kind of abstraction used for self and non-self elements, which we call the *universe* $\mathcal{U}$. For instance, negative selection algorithms have been implemented for both continuous and real-valued universes. In this paper, we only consider the universe $\mathcal{U} = \Sigma^L = \{0,1\}^L$, of binary strings of length L (larger alphabets can be encoded as binary). All existing negative selection algorithms based on the string universe suffer from a worst-case exponential size of D in the total size of the input [3]. This severely limits the practical applicability of negative selection [4]. We show here, however, that this is not an intrinsic problem of string-based negative selection, at least not when r-chunk or r-contiguous detectors are used.

The structure of this paper is as follows: In the next section, we define r-chunk and r-contiguous detectors and show that it is infeasible to enumerate all of them. The subsequent two sections show that an exhaustive enumeration of all detectors is not necessary: For both r-chunk and r-contiguous detectors, we can generate compressed representations of the entire sets that are themselves usable as detectors. In the last section, we summarize our findings and discuss the implications for related work.

2 String-Based Detectors and Detector Sets

Throughout the paper, we use the following notation conventions: Let $s \in \Sigma^L$ be a string. Then $|s| = L$ is the *length* of s and $s[i,\ldots,j]$ is the substring of s with length $j - i + 1$ that starts at position i. Let $S \subseteq \Sigma^L$ be a set of strings. Then $\overline{S} = \Sigma^L \setminus S$ is its *complement*.

Now we are ready to define what r-chunk and r-contiguous detectors are.

Definition 2.1 (Chunk detectors). *An r-chunk detector (d, i) is a tuple of a string $d \in \Sigma^r$ and an integer $i \in \{1,\ldots,L-r+1\}$. It matches another string $s \in \Sigma^L$ if $s[i,\ldots,i+r-1] = d$.*

Definition 2.2 (Contiguous detectors). *An r-contiguous detector is a string $d \in \Sigma^L$. It matches another string $s \in \Sigma^L$ if there is an $i \in \{1,\ldots,L-r+1\}$ with $d[i,\ldots,i+r-1] = s[i,\ldots,i+r-1]$.*

Definition 2.3 (Detector sets). *Given a self-set $S \subseteq \Sigma^L$ and an integer $r \in \{1, \ldots, L\}$, let* $\textsc{ChunkD}(S, r)$ *be the set of r-chunk detectors that do not match any string in S, and let* $\textsc{ContD}(S, r)$ *be the set of r-contiguous detectors that do not match any string in S.*

Figure 2 shows an example of a self-set together with its r-chunk detectors and r-contiguous detectors. This self-set is used several times as an example in the present paper.

01011 01010 01101	(000,1) (000,2) (000,3) (001,1) (001,2) (001,3) (100,1) (010,2) (100,3) (101,1) (011,2) (110,3) (110,1) (100,2) (111,3) (111,1) (111,2)	00000 10100 00100 10110 00110 10111 00111 11000 10000 11001 10001 11110 11111
Self-set S with $L = 5$ and $\lvert S\rvert = 3$	The set of 3-chunk detectors, $\textsc{ChunkD}(S, 3)$.	The set of 3-contiguous detectors, $\textsc{ContD}(S, 3)$.

Fig. 2. An example self-set $S \subseteq \Sigma^5$ together with the detector sets $\textsc{ChunkD}(S, 3)$ and $\textsc{ContD}(S, 3)$.

All previous attempts to build efficient algorithms for generating all r-chunk and r-contiguous detectors have resulted in a worst-case complexity that is exponential in the input size. For example, there is one algorithm for determining the size of $\textsc{ContD}(S, r)$ that runs linearly in $|S|$, but exponentially in r [5].

In fact, it is impossible to build an algorithm that generates *all* detectors and is guaranteed to run in less than exponential time. Consider the worst-case size of $\textsc{ChunkD}(S, r)$ and $\textsc{ContD}(S, r)$ in terms of the size of the input, that is, $|S| \cdot L$ (the space needed for storing r is negligible). Fix $r = L$, and let S contain only one arbitrary string s. That is, the input size is L. Now, the size of both $\textsc{ChunkD}(S, r)$ and $\textsc{ContD}(S, r)$ is $2^L - 1$: any string $d \in \Sigma^L \setminus \{s\}$ is both an r-contiguous detector, and an r-chunk detector at position 1. Hence, any algorithm that completely enumerates $\textsc{ChunkD}(S, r)$ or $\textsc{ContD}(S, r)$ must take exponential time in the worst case.

In principle, the infeasibility of generating all detectors might be due to some intrinsic hardness of the problem – maybe, as argued by Timmis et al. [6], it could be equivalent to an $\mathcal{NP}$-hard problem. However, we show in this paper that this is not the case. Instead, the problem is somewhat ill-posed. As an illustrating analogy, consider the task of enumerating all strings $s \in \Sigma^L$. This takes time 2^L, but it is certainly not very hard to do – the set of all strings of length L has a

trivial internal structure, which could be represented simply by the number L. We will show in the next two sections that efficient representations for both $\textsc{ChunkD}(S, r)$ and $\textsc{ContD}(S, r)$ can be constructed. The detection power of these representations is equivalent to that of the entire set, but the time and space required to construct them are guaranteed to be polynomial in the size of the input.

3 Negative Selection with Chunk Detectors

Our algorithm for r-chunk detection uses *patterns* that describe sets of several detectors at once. In the context of negative selection, the usage of patterns to describe complement sets was first explored by Esponda [7]. Our contribution in this section lies in applying this idea to r-chunk detection.

Definition 3.1 (Prefix patterns). *A prefix pattern π is a string over the alphabet $\Sigma \cup \{\diamond\}$ that has the form $\pi = vw$, where v is a string over Σ and w is a string consisting only of $\diamond$. A string s is described by $\pi = vw$ if v is a prefix of s and $|\pi| = |s|$. The set of all strings described by π is denoted by $P(\pi)$.*

For example, the pattern $01\diamond\diamond$ describes all binary strings of length 4 that start with 01. Obviously every set of strings of length L can be described by a set of prefix patterns, since every string is itself a prefix pattern. What we are interested in is a *minimal* representation of a set of strings in terms of prefix patterns. The following lemma, which is adopted from Esponda [7] with a slightly modified proof, shows that such a representation can be constructed efficiently.

Lemma 3.2 (Minimal prefix patterns for the complement of S). *Given a set $S \subseteq \Sigma^r$, denote by $\textsc{MinPrefix}(\overline{S})$ the smallest set of prefix patterns that describe exactly the strings in $\overline{S}$. $\textsc{MinPrefix}(\overline{S})$ is uniquely defined, has at most $O(|S|r)$ elements and can be constructed in time $O(|S|r^2)$.*

Proof. The set $\textsc{MinPrefix}(\overline{S})$ is constructed by the algorithm Construct-Minimal-Prefix-Patterns (Figure 3), which iterates over the set of all prefixes of all strings in S (we ignore the trivial case that S is empty). For each prefix, the last letter is flipped. If the resulting string p_i is not a prefix of any $s \in S$, then no prefix of p_i other than p_i itself has this property. Therefore, the patterns in the resulting set D describe pairwise disjoint sets of strings, and there is no pair (π_1, π_2) of two different patterns in D that could be replaced by a single pattern π_0 such that $P(\pi_1) \cup P(\pi_2) \subseteq P(\pi_0) \subseteq \overline{S}$. Hence, there is no pattern set smaller than D that describes the same strings.

To see that the entire set $\overline{S}$ is covered by the set D, fix an arbitrary string $\hat{s} \in \overline{S}$. Since $\hat{s}$ is not a prefix of any $s \in S$, there must be a shortest prefix $\hat{p}$ of $\hat{s}$ with the same property. Then the algorithm inserts $\hat{p} \diamond \ldots \diamond$, which describes $\hat{s}$, into the set D in line 6. Since there are $|S|r$ prefixes of the strings in S, the algorithm outputs at most $|S|r$ patterns. Its time complexity depends on the efficiency of the procedure used to check if the p_i match any string in S. Using a

suffix tree data structure (which can be constructed in time $O(|S|r)$ beforehand), this can be achieved in time $O(r)$ [8]. Hence, the resulting time complexity is $O(|S|r^2)$. □

For example, if $S = \{0101, 0011\}$, then algorithm CONSTRUCT-MINIMAL-PREFIX-PATTERNS yields $\{1\diamond\diamond\diamond, 011\diamond, 0100, 000\diamond, 0010\}$. This is indeed the minimal set of prefix patterns describing the complement of $\overline{S}$: No pattern can be removed from the set since some string in $\overline{S}$ would no longer be described, and no two patterns can be merged because the resulting pattern would describe strings that are not in $\overline{S}$.

Algorithm CONSTRUCT-MINIMAL-PREFIX-PATTERNS.
Input: A nonempty set $S \subseteq \Sigma^r$.
Output: The set MINPREFIX($\overline{S}$).

```
1  D ← ∅
2  for each s ∈ S do
3      for i = 1 to r do
4          p_i ← s[1, ..., i − 1]s[i]‾
5          if p_i is not a prefix of any s ∈ S then
6              D ← D ∪ {p_i ◇...◇}   (r − i)
7  output D
```

Fig. 3. Algorithm for constructing a minimal set of prefix patterns describing the complement of S.

Now we are ready to apply the idea of minimal prefix patterns to r-chunk detection. This is done by iterating over all positions $i \in \{1, \ldots, L - r + 1\}$, and considering only the substrings of the strings in S of length r at these positions. Let us denote this set by $S[i, \ldots, i + r - 1]$, then the set of all r-chunk detectors for position i is precisely the complement $\overline{S[i, \ldots, i + r - 1]}$. By generalizing the concept of r-chunk detectors to patterns, it is straightforward to construct an efficient representation of the set of all r-chunk detectors and to use it for r-chunk detection.

Definition 3.3 (Chunk patterns). *An r-chunk pattern is a tuple (π, i) of a prefix pattern π and an integer i. A string $s \in \Sigma^L$ is matched by (π, i) if there exists a string $d \in \Sigma^r$ described by π, such that the r-chunk detector (d, i) matches s.*

Algorithm EFFICIENT-CHUNK-NEGATIVE-SELECTION in Figure 5 provides an efficient approach for negative selection with r-chunk detectors. In lines 1 to 4 a set of r-chunk patterns D that describe exactly the r-chunk detectors of a self-set S is generated by using minimal prefix patterns. For the self-set from Figure 2, these r-chunk patterns are shown in Figure 4.

(1⋄⋄,1)	(100,2)	(00⋄,3)
(00⋄,1)	(111,2)	(11⋄,3)
	(0⋄⋄,2)	(100,3)
D_1	D_2	D_3

Fig. 4. This figure shows the set of 3-chunk patterns $D = D_1 \cup D_2 \cup D_3$ that the algorithm Efficient-Chunk-Negative-Selection constructs for the self-set from Figure 2. While the set of all r-chunk detectors can grow exponentially, the maximum size of the equivalent set of r-chunk patterns is polynomially bounded.

In lines 5 to 9 the set D is used to classify the elements from M. The algorithm correctly outputs for each element of M either self or non-self, since the set of r-chunk patterns is completely equivalent to the set of r-chunk detectors: If any given string is matched by an r-chunk detector, it is also matched by the pattern describing this detector; and if there is no r-chunk detector that matches a given string, then none of the equivalent r-chunk patterns match it.

By Lemma 3.2, constructing each D_i takes time $O(|S|r^2)$, and thus the time to construct D is $O(|S|r^2(L-r+1))$. Checking if a string matches a pattern takes exactly the same time as matching it against another string. Hence, we have reduced both time complexity of detector generation and the time complexity of detection itself from exponential to polynomial.

Algorithm Efficient-Chunk-Negative-Selection.
Input: A self-set $S \subseteq \Sigma^L$, an integer $r \in \{1, \ldots, L\}$ and a monitor set $M \subseteq \Sigma^L$.
Output: For each $m \in M$, either self or non-self for r-chunk detection.

```
1  for i = 1 to L − r + 1 do
2      S_i ← {s[i, ..., i + r − 1] | s ∈ S}
3      D_i ← {(π, i) | π ∈ MinPrefix(S̄_i)}
4  D = ⋃_i D_i
5  for each m ∈ M do
6      if m matches any pattern p ∈ D then
7          output "m is non-self"
8      else
9          output "m is self"
```

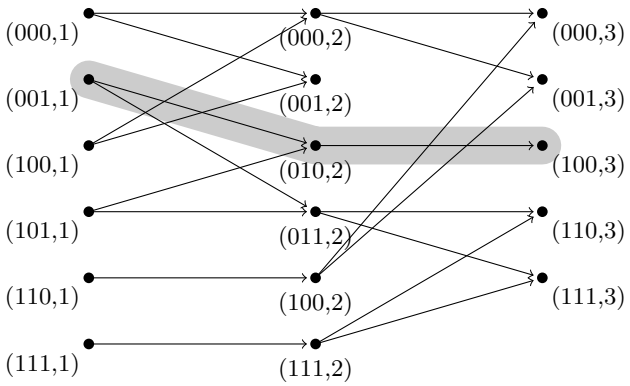
Fig. 5. Our efficient algorithm for negative selection with r-chunk detectors.

In the next section we present an efficient algorithm for r-contiguous detection using a slightly more complicated data structure that represents sets of contiguous detectors.

4 Negative Selection with Contiguous Detectors

In the previous section we used r-chunk patterns to describe sets of r-chunk detectors. In this section we introduce the *r-pattern graph* that is used to represent the r-contiguous detectors of a self-set $S \subseteq \Sigma^L$. The idea behind this data structure is that every r-contiguous detector can be constructed from a set of "overlapping" r-chunk detectors. As shown in Figure 6, we can construct a graph by linking every r-chunk detector to its overlapping neighbours. Now every r-contiguous detector corresponds to a path through this graph of length $L - r + 1$.

Of course, the simple procedure depicted in Figure 6 again suffers from an exponential number of nodes in the graph. Again, we can use patterns to reduce the number of nodes to polynomial. In the rest of this section, we prove that this compression procedure does not break the correctness of the construction. Also, it is not immediately obvious how the r-pattern graph can be used for self-nonself classification. This question will be addressed at the end of this section.



Derivation of the contiguous detector from the emphasized path:

3-chunk detectors	000
from the path:	010
	100
3-contiguous detector:	00100

Fig. 6. For the self-set from Figure 2 and $r = 3$, this figure shows how one can construct r-contiguous detectors from overlapping r-chunk detectors. The r-chunk detectors are arranged in levels and there is a directed edge from a detector d in level i to a detector d' in level $i + 1$ if $d[2, \ldots, r] = d'[1, \ldots, r - 1]$. Every path from the leftmost to the rightmost level of the graph corresponds to an r-contiguous detector.

Our compressed data structure, the r-pattern graph, is organized in $L-r+1$ levels from left to right and the vertices are labeled by r-chunk patterns. Edges are only drawn between vertices of consecutive levels. A path from the leftmost to the rightmost level represents a set of r-contiguous detectors and two different paths describe different sets of r-contiguous detectors.

Definition 4.1 (Pattern graphs). *Let $S \subseteq \Sigma^L$ be a self-set. The r-pattern graph for S is a directed graph with $L-r+1$ levels and constructed as follows:*

1. *Let $D = D_1 \cup \ldots \cup D_{L-r+1}$ be the set of r-chunk patterns for S, as generated by the algorithm in Figure 5. For every level i, construct $|D_i|$ vertices and label them bijectively with the patterns from D_i.*
2. *For every level i from $1, \ldots, L-r$, insert an edge from a vertex with label (π, i) to a vertex with label $(\pi', i+1)$ if $P(\pi[2, \ldots, r]) \supseteq P(\pi'[1, \ldots, r-1])$.*
3. *Repeatedly do the following:*
   (a) *Delete vertices from level 1 without outgoing edges.*
   (b) *Delete vertices from levels $2, \ldots, L-r$ without outgoing or incoming edges.*
   (c) *Delete vertices from level $L-r+1$ without incoming edges.*

The construction procedure for an r-pattern graph is directly given by its definition. Given the set D beforehand, which is constructed by the algorithm in Figure 5, the worst-case runtime for lines 1 and 2 is $O(r|D|^2)$ and $O(|D|^3)$ for line 3. Figure 7 shows an example of an r-pattern graph.

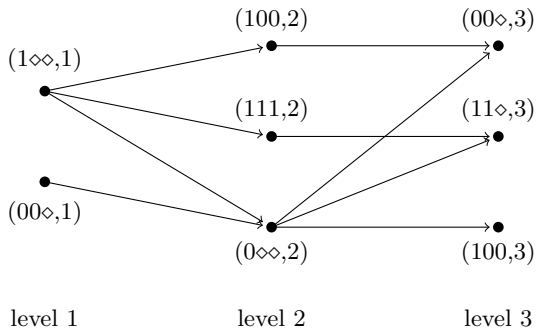
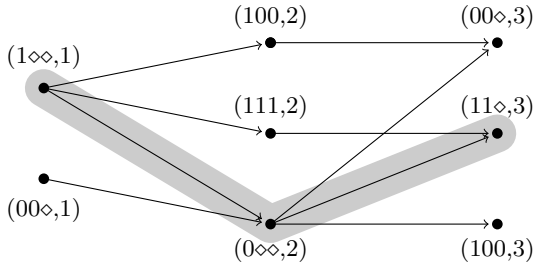


Fig. 7. The 3-pattern graph for the self-set S from Figure 2, which is a compressed version of the graph shown in Figure 6.

Definition 4.2 (Pattern paths). *Let $S \in \Sigma^L$ be a self-set and G its r-pattern graph. A path from a vertex in level 1 to a vertex in level $L-r+1$ is called a pattern path. It describes a string $d \in \Sigma^L$ if for every r-chunk pattern (π_i, i) on the path, the pattern π_i describes $d[i, \ldots, i+r-1]$.*

The role of pattern paths for r-contiguous detectors is similar to the role of r-chunk patterns for r-chunk detectors: (1) A pattern path describes r-contiguous detectors. These detectors can be constructed by merging the r-chunk patterns of the path according to their position and then filling up the $\diamond$ entries arbitrarily, as shown in Figure 8. Since for every position $i \in \{1, \ldots, L-r+1\}$ there exists an r-chunk pattern that matches the constructed string, it does not match any string in S and is, therefore, an r-contiguous detector. (2) Two pattern paths that differ in at least one vertex describe disjoint sets of r-contiguous detectors. (3) The pattern paths cover all r-contiguous detectors, as stated in the following lemma:



Merge of prefix patterns from the path:

Prefix patterns	1⋄⋄
from the path:	0⋄⋄
	11⋄
Merged path patterns:	1011⋄
Described detectors:	10110
	10111

Fig. 8. A pattern path in the 3-pattern graph from Figure 7 that describes two 3-contiguous detectors.

Lemma 4.3 (Pattern paths describe contiguous detectors). *Let $S \subseteq \Sigma^L$ be a self-set and G its r-pattern graph. The set of strings described by the pattern paths of G is exactly* $\textsc{ContD}(S, r)$.

Proof. We have discussed in the previous paragraph that every pattern path corresponds to ab r-contiguous detector. It remains to show that there is a pattern path for each $d \in \textsc{ContD}(S, r)$. Let $d \in \textsc{ContD}(S, r)$ be an r-contiguous detector. For every position $i \in \{1, \ldots, L-r+1\}$, the tuple $(d[i, \ldots, i+r-1, i)$

is an r-chunk detector and, hence, described by some r-chunk pattern $(\pi_i, i) \in D_i$. We show that G contains the path $(\pi_1, 1), \ldots, (\pi_{L-r+1}, L-r+1)$, which describes s by definition. Note, that it suffices to show that this holds after step 2 of the construction of the r-pattern graph in Definition 4.1. Consider the two patterns (π_i, i) and $(\pi_{i+1}, i+1)$ with $\pi_i = v_i w_i$ where v_i is a string over $\{0,1\}$ and w_i consists only of $\diamond$ (similarly $\pi_{i+1} = v_{i+1} w_{i+1}$ for appropriate v_{i+1} and w_{i+1}). If $|v_{i+1}| \geq |v_i| - 1$, then after step 2 there is an edge from (π_i, i) to $(\pi_{i+1}, i+1)$. The case $|v_{i+1}| < |v_i| - 1$ cannot occur for the following reason: By the construction of the r-chunk patterns in the algorithm in Figure 5, v_i is not a substring of any string from S at position i, but its prefix of length $|v_i| - 1$ is a substring of a string $s \in S$ at position i. Therefore, v_{i+1} is a substring at position $i+1$ in s, which contradicts the construction of the r-chunk patterns. □

Similar to the fact that pattern paths describe r-contiguous detectors, subpaths of them describe substrings of r-contiguous detectors. Thus, if we want to know whether a given string m matches an r-contiguous detector, it suffices to look at subpaths that represent substrings of length r. We use the following graph to look at possible paths for a string m and a position i:

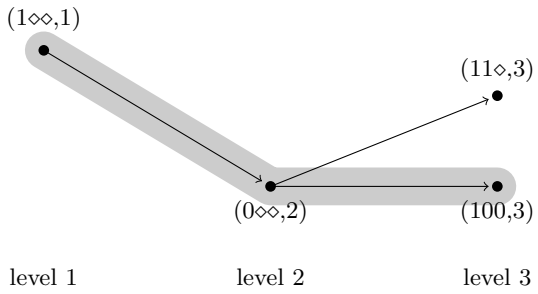
Definition 4.4 (Restricted pattern graph). *Let $S \subseteq \Sigma^L$ be a self-set, G its r-pattern graph, $m \in \{0,1\}^L$, and $i \in \{1, \ldots, L-r+1\}$. The (m,i)-restricted r-pattern graph for S is the induced graph of G that contains exactly the vertices (π, j) with $j \in \{i, \ldots, i+r-1\}$ where $\pi[1, \ldots, r+i-j]$ describes $m[j, \ldots, i+r-1]$.*

Given the r-pattern graph beforehand, a string m and an index i, the (m,i)-restricted graph can be constructed as in the definition, which needs time at most $O(|D|r)$. Figure 9 shows an example for restricted pattern graphs.

Lemma 4.5 (Contiguous detection with pattern graphs). *Let G' be the (m,i)-restricted r-pattern graph for a self-set $S \subseteq \{0,1\}^L$ and $m \in \{0,1\}^L$. The string m matches a detector $d \in \textsc{ContD}(S, r)$ at position i if, and only if, there is a path from the leftmost level (level i) to the rightmost level (level $\min\{i+r-1, L-r+1\}$) in G'.*

Proof. Let m match a detector $d \in \textsc{ContD}(S, r)$ at position i. By Lemma 4.3 we know that there exists a pattern path $(\pi_1, 1), \ldots, (\pi_{L-r+1}, L-r+1)$ in the unrestricted r-pattern graph that describes d. Since $m[i, \ldots, i+r-1] = d[i, \ldots, i+r-1]$, we also know that for every $j \in \{i, \ldots, i+r-1\}$, the prefix of length $r+i-j$ of π_j describes the substring of length $r+i-j$ at position j in m. Thus, there is a path from the leftmost to the rightmost level in G'.

For the other direction, consider a path from the leftmost to the rightmost level in the restricted graph. It is part of a pattern path in the unrestricted r-pattern graph. We choose an r-contiguous detector d from the set of detectors described by the pattern path, such that the free entries between position i and $i+r-1$ (the positions with only $\diamond$ entries in all r-chunk patterns) are filled like in m. Then d matches m at position i since the construction of the restricted graph assures that, whenever the considered pattern path forces a 0 (or 1) entry in a position $j \in \{i, \ldots, i+r-1\}$, m already has a 0 (or 1) entry at that position. □



Path describes a substring of length 3
of a detector:

Merged path patterns:	10100
Monitor string m:	10101
Matching detector:	10110

Fig. 9. The $(10101, 1)$-restricted 3-pattern graph for the self-set from Figure 2 with a path from level 1 to 3.

Given an (m, i)-restricted r-pattern graph, the property of the last lemma can be tested as follows: Assign to each vertex of level i the weight 1. Then iterate through all of the remaining levels and set the weight of every vertex to the maximal weight of a predecessor vertex plus one. There is a path from the leftmost to the rightmost level if the weight of a vertex in the rightmost level equal the number of levels in the restricted graph. This procedure takes time at most $O(|D|^2 r)$.

Bringing it all together, the algorithm in Figure 10 correctly detects for each element of a monitor set M, whether it is self or non-self. The algorithm uses time at most $O(|S|Lr^2)$ for the construction of the set D in line 1 and $O(|S|Lr)$ is an upper bound for the number of patterns in D. In line 2 the running time for the construction of the r-pattern graph is at most $O(|D|^2 r + |D|^3)$. For the detection of a single element m from the monitor set, the algorithm iterates over all possible positions $i \in \{1, \ldots, L - r + 1\}$. For each position, it constructs the (m, i)-restricted pattern graph and decides whether there is a path from the leftmost to the rightmost level. Thus, the time for the detection of a single element is at most $O(|D|^2 Lr)$.

5 Discussion

We have seen that negative selection based on r-chunk and r-contiguous detector is inefficient if *all* detectors are generated. However, the complete detector sets are highly redundant. Our results show that equivalent, but significantly smaller representations of these sets can be constructed. It is therefore unnecessary to

Algorithm Efficient-Contiguous-Negative-Selection.
Input: A self-set $S \subseteq \Sigma^L$, an integer $r \in \{1, \ldots, L\}$ and a monitor set $M \subseteq \Sigma^L$.
Output: For each $m \in M$, either self or non-self for r-contiguous detection.

```
1  construct the set D of r-patterns
2  construct the r-pattern graph G from D
3  for each m ∈ M do
4      for i = 1 to L − r + 1 do
5          construct the (m, i)-restricted r-pattern graph G′
6          if there is a path from the leftmost to the rightmost level in G′ then
7              output m is "non-self"
8      if m was not detected as "non-self" then
9              output m is "self"
```

Fig. 10. Our efficient algorithm for negative selection with r-contiguous detectors.

enumerate all detectors explicitly. In the case of r-chunk detectors, our construction is a straightforward continuation of Esponda's results [7] and should not be much harder to implement than the original procedure. On the other hand, the construction for r-contiguous detectors is still a little more involved. We would like to point out here that our main focus here was to show that r-contiguous detection is *principally* feasible in polynomial time. For concrete implementations, the algorithm developed in the previous section can still be streamlined.

It is important to put our work in context with the work carried out by Stibor [3, 6, 9], who demonstrated that the problem of deciding if any r-contiguous detector is generable for a given self-set can be expressed as an instance of the k-CNF satisfiability problem (SAT), which is $\mathcal{NP}$-complete. This result is not a contradiction to our findings. Formally, Stibor showed that the generability problem for r-contiguous detectors – let us denote it GenP – is polynomial-time reducible to SAT, and this implies GenP $\in \mathcal{NP}$. However, it does not imply that GenP is $\mathcal{NP}$-complete – to prove that, one needs additionally a reduction in the opposite direction. In fact, our results imply GenP $\in \mathcal{P}$: Given S and r, construct the r-pattern graph G. Detectors are generable if, and only if, there is a pattern path in the graph. This test takes polynomial time. Hence, it is very unlikely that detector generability is complexity-wise equivalent to boolean satisfiability, since this would now imply $\mathcal{P} = \mathcal{NP}$.

In summary, our work shows that string-based negative selection is a computationally feasible approach. It remains to be seen if this finding leads to a substantial improvement to the real-world performance of negative selection, or even the desired "killer application" for AIS. However, since the negative selection algorithm has been a source of inspiration for many in the past, maybe there is now a little more hope.

References

1. Forrest, S., Perelson, A.S., Allen, L., Cherukuri, R.: Self-nonself discrimination in a computer. In: Proceedings of the IEEE Symposium on Research in Security and Privacy. (1994) 202–212
2. Forrest, S., Hofmeyr, S.A., Somayaji, A.: Computer immunology. Communications of the ACM **40** (1997) 88–96
3. Stibor, T.: Foundations of r-contiguous matching in negative selection for anomaly detection. Natural Computing (2008)
4. Stibor, T.: On the Appropriateness of Negative Selection for Anomaly Detection and Network Intrusion Detection. PhD thesis, Darmstadt University of Technology (2006)
5. D'Haeseleer, P.: An immunological approach to change detection: theoretical results. In: Proceedings of the 9th IEEE Computer Security Foundations Workshop. (1996) 18–26
6. Timmis, J., Hone, A., Stibor, T., Clark, E.: Theoretical advances in artificial immune systems. Theoretical Computer Science **403** (2008) 11–32
7. Esponda, C.: Negative Representations of Information. PhD thesis, University of New Mexico (2005)
8. Ukkonen, E.: On-line construction of suffix-trees. Algorithmica **14**(3) (1995) 249–260
9. Stibor, T.: Phase transition and the computational complexity of generating r-contiguous detectors. In de Castro, L., von Zuben, F., Knidel, H., eds.: ICARIS 2007: 6th International Conference on Artificial Immune Systems. Volume 4628 of Lecture Notes in Computer Science., Springer (2007) 142–155