Generating optimal repertoire of antibody strings in an artificial immune system

Sławomir T. Wierzchoń

Institute of Computer Science, Polish Academy of Sciences

01-267 Warszwa, ul. Ordona 21

e-mail: stw@ipipan.waw.pl

***Abstract**: In this paper an idea of the artificial immune system (or AIS for brevity) is explained. Restricting to so-called binary AIS, methods for generating a repertoire of lymphocytes of minimal size are reviewed and new algorithm of low space complexity is discussed. Besides, recipes for counting so-called holes, as well as counting the total number of unrecognizable strings are given.*

Keywords: Artificial Immune Systems, Anomaly Detection, Receptors Generation

1. Introduction

Careful analysis of biological processes and phenomena has led to many interesting computer applications. Reflecting on fundamental principles of animal behaviour, Norbert Wiener proposed cybernetics - a general framework for studying different aspects of communication and control in living organisms. Imprecise nature of human reasoning and perception gave an impulse to fuzzy sets theory initiated by Lotfi Zadeh. The works by Warren Mc Culloch focused on mathematical formalism describing the behaviour of a single neurone resulted in Rosenblat's perceptron and next, in highly elaborated neural networks. John Holland's genetic algorithms are a resemblance of the basic idea of evolution formulated by Darwin and Lamarck. Molecular computing, initiated by Leonard Adleman, transfers evolutionary concepts into their roots: DNA strands compete in artificially prepared environment and solve complex problems.

Immunology is another branch of biological sciences. Starting from early 70's it becomes a subject of growing interest of mathematicians, physicists and computer scientists. Perelson and Weisbuch in [18] provide an exhaustive review of different models of natural immune system (NIS) as well as a historical perspective of the researches in this field, while edited by Dasgupta volume [6] is devoted to computer-science oriented applications. The main problem the NIS cope with is that of distinguishing between *self* and *non-self* (called pathogens or *antigens*) proteins. From a computer science perspective the NIS is a complex, self organizing and highly distributed system that has no centralized control and uses learning and memory when solving particular, complex, tasks. The learning process does not require negative examples and the acquired knowledge is represented in explicit form. Such a system has the advantages of classifier systems, [7], neural networks and case-based retrieval, 13]. One should take into account, however, that the field of Artificial Immune Systems (or, AIS) is not a well established discipline. It is too young, and even there is a lack of definite models and theories of the NIS. Nevertheless, it affords interesting metaphor according to which a problem to be solved is treated as an *antigen* and a solution to the problem as an *antibody* (i.e. a protein that detects and destroys the antigen). This metaphor, has found interesting implementations, like: computer virus detection [8], monitoring UNIX processes [9], anomaly detection in time series [5], fault analysis [16], process diagnosis [17], numerical optimization [1], recognizing promoters in DNA sequences [14] or mortgage fraud detection [15].

In this paper we focus on the problem of generating a repertoire of nonredundant antibodies (termed here *receptors*) that can be used in detecting anomalies in a given data set. Here both self and non-self proteins are represented as binary strings of fixed length. This binary representation is commonly used in the literature, cf. [7], [8]. After introducing the method of receptors activation (Sect. 2.1) and defining building blocks for receptors generation (Section 2.2) various methods of constructing the repertoire are discussed (Section 2.3). Section 3 introduces a new algorithm generating complete repertoire of receptors. In Sect. 3.1 a graphical representation of the receptors is proposed. Using this representation, in Sect. 3.2, a simpler method to count so-called holes is given and its generalization to counting the number of all unrecognizable strings from a given strings universe is presented. By the way we obtain a startling result: the set of self strings blocks some templates making impossible to construct some additional (apart of holes) receptors. This means that (at least in small immune systems) we cannot make the probability of detecting an intrusion arbitrarily small (an assumption commonly used in the literature). Section 3.2 is of great practical importance: the algorithms presented here allow to find *in advance* the number of receptors and to determine their quality expressed in the number of unrecognizable strings. Finally, an effective (of low space complexity) algorithm for generating nonredundant receptors repository is presented in Section 3.3.

2. Anomaly detection

The idea of using immunological principles in anomaly detection was proposed in [10]. It can be expressed as the problem of distinguishing *self* from *nonself*, where by *self* we understand all typical ("normal") patterns of activity of a system (or process) or a collection that we wish to protect or monitor and *nonself* is all irregular (or unacceptable) patterns. To solve this problem Forrest and her co-workers developed in [10] a negative-selection algorithm imitating the behaviour of T-lymphocytes in the NIS. It consists of four main steps:

1. Define S as a collection of self patterns.
2. Generate a set of receptors R, such that each $r \in R$ doesn't recognize any self pattern, $s \in S$.
3. Apply R to monitoring data set D.
4. If at least one receptor activates when analyzing information $d \in D$, it means that d manifests anomaly.

The algorithm has a number of features which distinguish it from most other methods (cf. [3] for details): it doesn't require any prior knowledge of anomalies, it uses incomplete but tunable (by varying the threshold τ) detection, and the detection scheme is inherently distributable, i.e. small sections of the protected system can be checked separately by independently created receptor sets. To implement the algorithm we should define: (a) patterns and receptors representation, (b) the method of their activation, (c) the method of receptors generation, and possibly, (d) algorithm efficiency.

In the sequel we will focus on the simplified binary representation of the patterns introduced by Farmer, Packard and Perelson in [7]. That is we will assume that U is the set of all binary strings of length λ; obviously it has $|U| = 2^{\lambda}$ different strings. Both S and R are subsets of U, and the elements of U - S represent antigens. Having defined receptor representation we can discuss the problems of their activation and generation. This is the subject of succeeding subsections.

2.1 Receptors activation

In the NIS lymphocytes, treated here as receptors, detect pathogens by binding to them. Receptors can only be activated by a pathogen, a member of U - S, if the receptors similarity (affinity in biological nomenclature) for the pathogen exceeds a certain affinity threshold. Affinity is computed in a way reminiscent from the classical *lock and key* metaphor: two molecules bind if they have complementary shapes. In our framework, the process of a receptor's activation can be defined as follows. Suppose that r and x are two strings such that the former represents receptor and the later represents a protein. We introduce a Boolean function `match`(r,x) such that `match`(r,x) = `FALSE` if $x \in S$ (i.e. s is a self string) and `match`(r,x) = `TRUE` if x is an antigen.

There is no unique definition of the `match`(r,x) function. In the simplest case it can be the Hamming distance between both strings, but here we will use τ-contiguous-bits rule proposed in [11]: `match`(r,x) = `TRUE` only if r and x are identical on at least τ contiguous positions, where τ is a given threshold value. If, for instance, $\lambda = 6$, $\tau = 3$ and the strings r (receptor) and x_1, x_2 are of the form r = `000110`, x_1= `001100` and x_2= `000100`, then `match`(r,x_1) = `FALSE` while `match`(r,x_2) = `TRUE`, so x_1 is a self pattern and x_2 is an antigen. Other matching rules are discussed in [7] and [14].

2.2 Templates

The idea of a template was introduced in [11]. Roughly speaking, templates are building blocks from which the receptors are constructed.

Let w be a binary string of length τ (τ is the threshold value mentioned in Sect. 2.1). We will consider strings of length λ over the alphabet {0,1,*} where * stands for "irrelevant"[1]. By a template $t_{i,w}$ of order τ, we understand a string (of length λ), whose substring of length τ taken from position i equals w, and all the remaining positions of the template are filled by the * symbol. In genetic algorithms terminology (consult e.g. [12]) a template of order τ is a schema of order[2] τ in which all the significant bits are contiguous. For instance, when $\lambda = 6$, $\tau = 3$, w = `010` then $t_{1,w}$ = `010***`, $t_{2,w}$ = `*010**`, $t_{3,w}$ = `**010*`, and $t_{4,w}$ = `***010`.

The set of all possible templates, denoted T, contains $(\lambda - \tau + 1)\cdot 2^{\tau}$ different elements. We split T into two disjoint subsets: T_S consisting of all the templates contained in at least one self string and the set of remaining templates, T_R, used to construct receptor strings. Typically T_S is a relative small fraction of T. Following [2] we will naively[3] represent the set T as the matrix T with 2^{τ} rows and $(\lambda - \tau + 1)$ columns: $\mathsf{T}[w,i] = 0$ if $t_{i,w} \in T_S$ and $\mathsf{T}[w,i] = 1$ if $t_{i,w} \in T_R$.

Example 1: Let $\lambda = 10$, $\tau = 4$. Given the set of twelve self strings shown in the leftmost column of Table 1, the sets T_S (of self templates) and T_R (of non-self templates) are represented by the table T consisting of 16 rows and 7 columns shown below.

2.3 Existing algorithms for receptors generation

Working with a binary AIS we face the next problem: Having the set S of self strings, generate the set R containing as small as possible receptors which recognize all the antibodies from $U - S$.

---

1 It is possible to consider strings over an alphabet Σ of m symbols, however we restrict our analysis to the binary systems.

2 The order of a schema is defined as the number of relevant positions in this schema. For instance if x_1 = 0000**** and x_2 = 00000*** then $order(x_1) = 4$ and $order(x_2)=5$.

3 Linked lists or sparse arrays are more efficient representations. We use the matrix representation for its illustrative power only.

S	no	w	T[w,1]	T[w,2]	T[w,3]	T[w,4]	T[w,5]	T[w,6]	T[w,7]
0011000000	0	0000	0	0	1	1	0	0	0
0011010000	1	0001	0	1	0	1	1	1	1
0011110000	2	0010	1	0	1	0	1	1	1
0000010000	3	0011	0	0	1	1	1	1	1
0001010000	4	0100	1	1	0	1	0	1	1
0001000000	5	0101	1	1	0	1	1	1	1
0001110000	6	0110	1	0	1	1	1	1	1
1000010000	7	0111	1	0	0	1	1	1	1
1100010000	8	1000	0	0	1	0	0	0	1
1101000000	9	1001	1	1	1	1	1	1	1
1111000000	10	1010	1	0	1	0	1	1	1
0011100000	11	1011	1	1	1	1	1	1	1
	12	1100	0	1	0	0	0	1	1
	13	1101	0	1	0	1	1	1	1
	14	1110	1	0	0	0	1	1	1
	15	1111	0	1	0	1	1	1	1

Table 1. Matrix T representing the set T_S (of self) and T_R (of nonself) templates

A naive solution of this problem is to generate randomly candidate receptors and then test (*censor*) them to see if they match any self string. If a match is found, the candidate is rejected. This process is repeated until the desired number of receptors is generated. This algorithm resembles the way in which T-cells are recruited in the thymus. It is ineffective, however, because the receptors grow exponentially with the size of self. Denoting P_m the probability that two random strings match at least τ contiguous positions and P_f - the probability that the set of receptors fail to detect an antigen, the time complexity of this algorithm is $O(-\ln(P_f)\cdot|S|/(P_m\cdot(1-P_m)^{|S|})$ and the space complexity is $O(\lambda\cdot|S|)$, cf. [4] for details.

Helman and Forrest proposed in [11] a more efficient algorithm which runs in linear time with the size of self (and receptors). It consists of two stages. First, the set of templates from which receptors can be constructed is identified, and numbering of the templates is established. Next, this numbering is used to construct randomly the receptors. Unfortunately, this way we obtain many redundant receptors. In [2] a greedy algorithm based on the same principles was proposed; it generates better coverage of the string space by placing detectors as far apart as possible. However, its space complexity is of order $O((\lambda - \tau)^2\cdot 2^\tau$.

3. New algorithm for generating receptors

3.1 Graphical representation of the receptors

Suppose that T_R consists of n templates $t_{1,w}$ (n=9 in Example 1). In [19] it was observed that we can construct n binary trees, whose roots are just these templates. A template $t_{i+1,v}$ is a child of $t_{i,w}$, $1 \le i \le (\lambda - \tau + 1)$, only if both the templates are identical on the positions i+1,.., i+τ-1. Having such a binary tree we read off subsequent receptors by: (a) fixing a path from the root to a leaf, (b) copying the substring w from the template $t_{1,w}$ to first τ positions of the receptor, and finally, copying on (τ+i-1)-th position, $1 \le i \le (\lambda - \tau + 1)$, the last significant bit from the templates $t_{i+1,w}$. To be more concrete consider the next example studied in [2].

Example 2: Suppose that S consists of ten strings: `001100`, `001101`, `001111`, `010001`, `010101`, `011100`, `011111`, `100001`, `110001`, `110100`, that is λ = 6 and τ = 3. The sets T_S and T_R are shown in Table 2 (given later) and Figure 1 shows the binary trees that represent seven receptors: r_1 = `000110`, r_2 = `101000`, r_3 = `101010`, r_4 = `101011`, r_5 = `111000`, r_6 = `111010`, r_7 = `111011`.

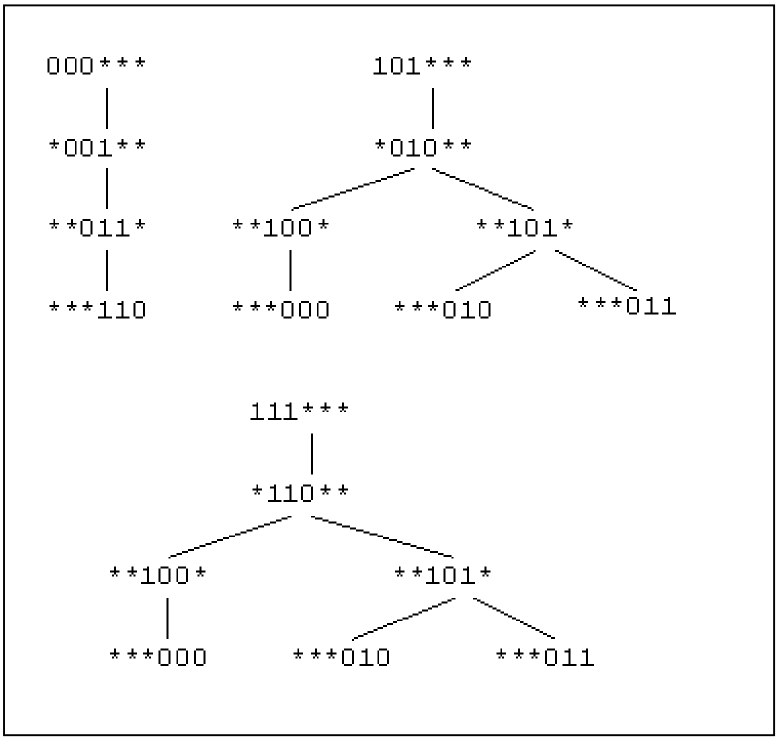

Fig. 1 Binary trees representing receptors from Example 2

Fig. 2 shows a simple procedure allowing generating complete repertoire of the receptors. Here we use the next terms: partial receptor is a string of length $l < \lambda$ obtained by concatenating templates $t_{1,w}, ..., t_{i,v}$ where $i = l - \tau + 1$. If p is a partial receptor of length l its left child is the template $t_{i+1,z}$ where the last character in z is 0 and its right child is the template $t_{i+1,z'}$ where the last character in z' is 1.

```
Procedure IdentifyReceptors
begin
   Identify all the templates t_{1,s} ∉ T_S and write them on a file, say file1. Call a
   member of file1 a partial receptor (initially its length is just τ).
   while (length of partial receptor p is less than λ) do
   begin
      while not (end of file1) do
         Read a partial receptor, p, and check if the left and right child exists. If
         the left child exists write on file2 the string p+'0'; if the right child
         exists write on file2 the string p+'1'.
      end {of while}
      Rename file2 as file1.
   end {of while}
end
```

Fig. 2 A pseudocode of the procedure generating full set of receptors

This procedure is ineffective because of two main reasons: (a) there exist partial receptors having no children, and (b) the number of *efficient* receptors is actually smaller than that computed by `IdentifyReceptors`. To explain the last drawback, let us examine Fig. 1. The templates $t_{2,\texttt{010}}$ and $t_{2,\texttt{110}}$ belonging to different trees have common children or more precisely, common subtrees: one rooted at $t_{3,\texttt{100}}$ and the second at $t_{3,\texttt{101}}$. Hence the trees from Fig. 1 can be represented more compactly by the structures shown in Fig. 3.

Receptors r_3, r_4, r_6 and r_7 contain the template $t_{3,\texttt{101}}$. It is easy to note that the strings containing the template $t_{1,\texttt{101}}$ are recognized by the receptors r_3 and r_4, while the string containing the template $t_{1,\texttt{111}}$ are recognized by r_6 and r_7. Similarly, the receptors containing the template $t_{4,\texttt{010}}$ are recognized by r_3 and r_6, and receptors containing the template $t_{4,\texttt{011}}$ - by r_4 and r_7. Analyzing the "neighbourhood" of the template $t_{3,\texttt{101}}$ we state that the strings that agree with the schema `1*10**` are detected by the receptors r_3, r_4 if the second position equals `0` or r_6, r_7 if the second position equals `1`. Similarly, the strings containing `***01*` are recognized by the receptors r_3, r_6 if the sixth position equals `0` or r_4, r_7 if the sixth position equals `1`. This shows that constructing an effective set of receptors, we must choose (in common subtrees) path ending on different leaves. In our case the effective set of receptors is of the form $\{r_1, r_2, r_3, r_7\}$ or $\{r_1, r_2, r_4, r_6\}$ or $\{r_1, r_3, r_5, r_7\}$ or $\{r_1, r_4, r_5, r_6\}$.

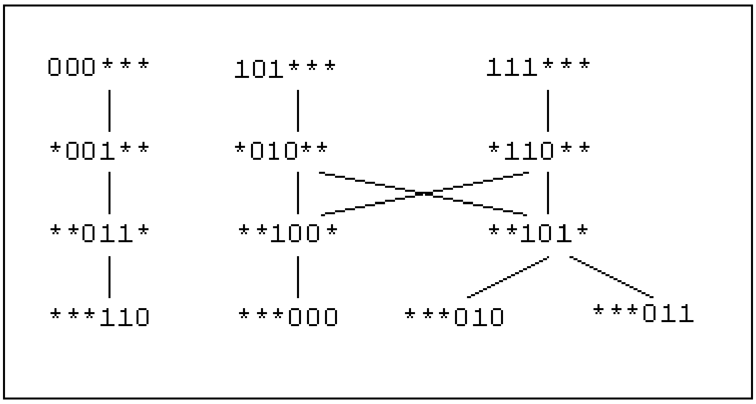

Fig. 3 Compact representation of the receptors from Fig. 1

In the real problems situation slightly complicates. Fig. 4 presents a typical situation: in the original tree (cf. Example 1 with λ=10, τ=4) common nodes for different paths have been identified. An interested reader can verify that in this situation the minimal set of receptors is obtained by identifying 12 paths starting from the root `0010******` and ending in the leaves[4]. Theorem 1, proved in [20] summarizes these observations and gives a hint how to create the set of nonredundant receptors.

Theorem 1. Let j be a fixed integer such that $1 \le j < \lambda\text{-}\tau\text{+}1$. Let p_1 and p_2 be two partial receptors of length τ+j-1 with identical τ-1 last characters. Denote these characters as *tmp*. If *tmp*+'0' and *tmp*+'1' are valid templates, then the set of full length receptors obtained from the four receptors p_1+'0', p_1+'1', p_2+'0' and p_2+'1' has the same discriminative power as the set of receptors obtained from the partial receptors {p_1+'0', p_2+'1'} or { p_1+'1', p_2+'0'}.

Concerning the first drawback, consider the set T_R from Example 1 and assume we want to create receptors starting from the template $t_{1,0111}$. Its children are $t_{2,1110}$ (left child) and $t_{2,1111}$ (right child). Since $t_{2,1110} \in T_S$, we can search for the children of $t_{2,1111}$ only. But both the children of $t_{2,1111}$ are members of T_S, what means that the templates $t_{1,0111}$ and $t_{2,1111}$ should be excluded from T_R. The problem of how to reduce the original set T_R is discussed in the next section.

[4] To simplify the picture, the star symbols are omitted. One should remember that the template on i-th level has (i-1) leading stars and (λ-τ-i+1) stars after the last significant position.

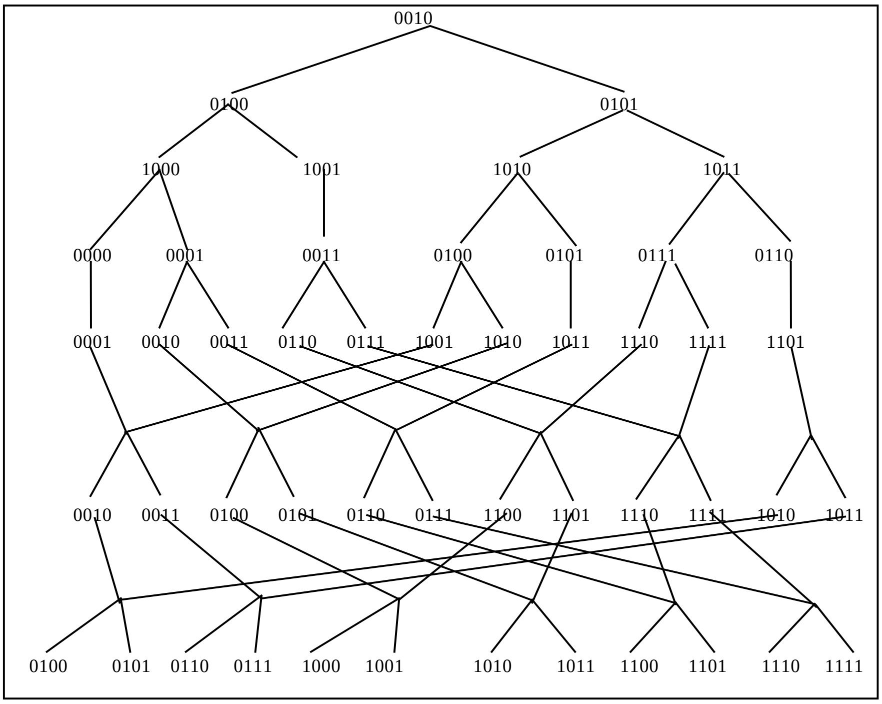

Fig. 4 More realistic representation of the set of detectors

3.2 Counting the set of nondetectable strings.

D'haeseleer noted in [2] that the set of all possible receptors is not capable of recognizing all non-self strings. Particularly, if S contains two strings s_1 and s_2 that match each other over (τ-1) contiguous bits, they may induce two other strings h_1 and h_2 that cannot be detected because any candidate detector would also match either s_1 or s_2; h_1 and h_2 are said to be *holes*. This can be proved using the arguments similar to these exploited in previous subsection.

More precisely, suppose that S contains two strings $s_1 = w_1w_2w_3$, $s_2 = w_4w_2w_5$, where the length of substring w_2, denoted λ_2, is $(\tau\text{-}1) \le \lambda_2 \le (\lambda\text{-}2)$, substrings w_1 and w_4 are of length $1 \le \lambda_1 \le (\lambda\text{-}r)$, and substrings w_3 and w_5 are of length $1 \le \lambda_3 \le (\lambda\text{-}\tau)$, and $\lambda_1 + \lambda_2 + \lambda_3 = \lambda$. Then the strings $h_1 = w_4w_2w_3$, $h_2 = w_1w_2w_5$, belong to the set U - S, but cannot be detected by any of the receptors. To illustrate this phenomena, consider the set of self strings from Example 2, i.e. λ = 6 and let τ = 3. Let us choose two strings s_1 = `00`**`11`**`01`, s_2 = `01`**`11`**`00`. We observe that the substring w_2 has the form `11`. Hence the holes are: h_1 = `011101`, h_2 = `001100`. The string h_1 contains the next schemas: {`011***`, `*111**`, `**110*`,

***101}. The first three schemas are also present in the self string s_2, whereas the last schema is present in the string s_1. This means that each schema used to detect h_1 is contained in the string s_1 or s_2.

To determine the number of holes we can use again the reasoning from previous section. That is we construct a graphical structure, but this time we focus on the templates from the set T_S. Figure 5 presents such a structure for the set S defined above.

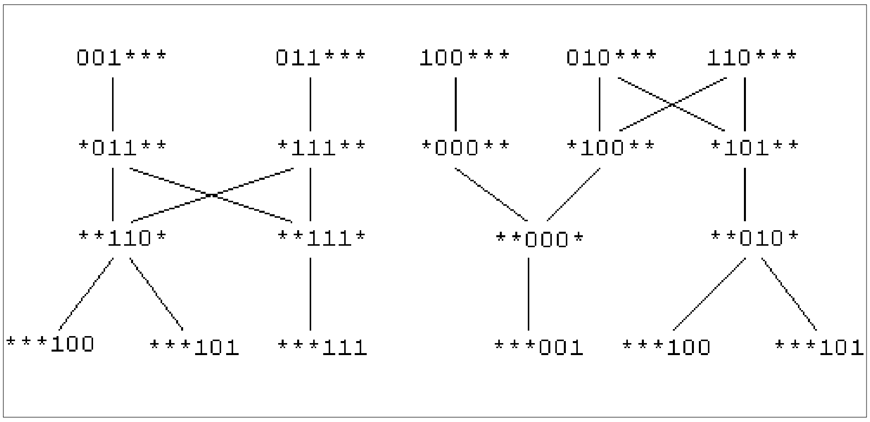

Fig. 5 Graphical representation of the set of holes generated by the set S from Example 2.

To generate the holes we can use the procedure GenerateReceptors with minor modification: this time we use only the templates from the set T_S. Since we search for a maximal number of paths from the roots to the leaves, hence we can apply the method proposed in [11]. First we define the table H with entries H[w,i] = 1 - T[w,i][5]. Now if H[w,i] $\neq 0$ (i.e. the template $t_{i,w} \in T_S$) we modify its value according to the equation

H[w,i] = H[LeftParent(w),i-1 + H[RightParent(w),i-1]

for $i = 2,...,\lambda-\tau+1$. For instance if w = 101 then LeftParent(w) = 010 and RightParent(w) = 110. Table 4 shows the result of using this procedure. Summing up the values from the last column we find that the number of holes equals 13. Note however that, according to Fig. 4, this number includes self strings. Hence the number of strict holes induced by strings from S is 13 - 10 (the cardinality of S) = 3.

[5] This method is more effective than that proposed in [2]. Particularly, when the algorithm is implemented with linked lists, its space complexity is $O(|S|\cdot(\lambda-\tau+1)$ where $|S|$ stands for the cardinality of S.

The existence of holes is not quite negative phenomena. They can be treated as manifestation of an “almost” normal behaviour. Besides, it was proved in [3] that all practical matching rules with a constant matching probability (i.e. each detector matches equally many strings and vice vers) can exhibit holes.

$t_{1,w}$	H[w,1]	$t_{2,w}$	H[w,2]	$t_{3,w}$	H[w,3]	$t_{4,w}$	H[w,4]
000***	0	*000**	1	**000*	3	***000	0
001***	1	*001**	0	**001*	0	***001	3
010***	1	*010**	0	**010*	2	***010	0
011***	1	*011**	1	**011*	0	***011	0
100***	1	*100**	2	**100*	0	***100	4
101***	0	*101**	2	**101*	0	***101	4
110***	1	*110**	0	**110*	2	***110	0
111***	0	*111**	1	**111*	2	***111	2

Table 2. Counting the number of holes

It appears that the number of unrecognizable strings is in general greater than that computed by the procedure described above. The reason is such that T_R contains inefficient templates. A template $t_{i,w} \in T_R$ is inefficient if either its children or parents belong to T_S. D’haeseleer in [2] proposed a solution to this problem manipulating on the set T_R. Here we propose another solution using only the set T_S which is much smaller than T_R. The procedure `FindIneffective` shown in Fig. 6 solves this problem. Its effect on the initial set T_R given in Example 1 is shown in Table 3 (new members of T_S are in bold).

Since the set T_R consists now of the effective templates only, counting holes generated by the enriched set T_S we are able to find total number of unrecognizable strings in U given the set S of self strings. In our case the number of unrecognizable strings equals actually 96.

3.3 Generating effective receptors

To identify a subset of relevant receptors in the set of all receptors we must extend the main while-loop of the procedure from Fig. 2. Suppose we have constructed a partial receptor (its length is smaller than λ) and by means of while-step we must merge it with its left and/or right child. If it has only one child, we precede as before. Otherwise, we make use of Theorem 1, that is we must check if current j-th template belongs to more that one tree. To do so we store current templates on a linked list *redundant*. If a current template belong to the list it means that this is the case and we should choose only one path in each tree. The full idea is explained on Fig. 7.

Procedure `FindIneffective`;
begin
  for i:=2 **to** (λ-τ+1) **do**
  for all $t_{i,w} \in T_S$ **do**
  begin {remove parents whose children belong to T_S }
    identify parents of the child $t_{i,w}$; if all the children of the parents belong to T_S, move the parents to T_S.
  end;
  for i:=1 **to** (λ-τ) **do**
  for all $t_{i,w} \in T_S$ **do**
  begin {remove children whose parents belong to T_S }
    find the children of $t_{i,w}$ and check if the second parent also belongs to T_S ; if so, mark both the children as members of T_S..
  end;
  for i:=(λ-τ+1) downto 2 **do**
  for all $t_{i,w} \in T_S$ **do**
  begin
    identify parents of the child $t_{i,w}$; if all the children of the parents belong to T_S, move the parents to T_S.
  end;
end; {`FindIneffective`}

Fig. 6 Pseudocode for finding efficient set T_R

S
0011000000
0011010000
0011110000
0000010000
0001010000
0001000000
0001110000
1000010000
1100010000
1101000000
1111000000
0011100000

no	w	T[w,1]	T[w,2]	T[w,3]	T[w,4]	T[w,5]	T[w,6]	T[w,7]
0	0000	0	0	**0**	1	0	0	0
1	0001	0	**0**	0	1	1	**0**	**0**
2	0010	1	0	1	0	1	1	**0**
3	0011	0	0	1	1	1	1	**0**
4	0100	1	1	0	1	0	1	1
5	0101	1	1	0	1	**0**	1	1
6	0110	1	0	1	1	1	1	1
7	0111	**0**	0	0	1	1	1	1
8	1000	0	0	1	0	0	0	1
9	1001	**0**	1	1	**0**	1	**0**	1
10	1010	1	0	1	0	1	1	1
11	1011	**0**	1	1	**0**	1	1	1
12	1100	0	1	0	0	0	1	1
13	1101	0	1	0	1	1	1	1
14	1110	1	0	0	0	1	1	1
15	1111	0	**0**	0	**0**	1	1	1

Table 3. The effect of the procedure `FindIneffective` applied to the data from Table 3 (new ineffective templates are in bold)

To illustrate the effectiveness of the algorithm two dataset were taken: one consisting of 12 self strings of length 10 (given in Example 1) and the second consisting of 20 random strings of length 12. Table 4 shows the result of the algorithm given in Fig. 2 and its modified version from figure 7. We observe that an increase of 1 in τ changes the number of receptors almost two times. This follows from the fact that a single receptor recognizes $2^{\lambda-\tau-1}\cdot(\lambda-\tau+2)$ different strings[6], [20]. Hence the number of receptors grows approximately as $2\cdot(\lambda-\tau+2)/(\lambda-\tau+1)$

Interestingly, we can use the procedure `FindIneffective` to study the effectiveness of the set of receptors. Namely, we construct the set T_S by adding to it the templates that are not present in any receptor. Then, for instance we can count fast and easily the number of unrecognized strings by any subset of the set of receptors. Table 4 shows results of this procedure - cf. columns **c**,**d**,**g**, and **h**.

Procedure `IdentifyNonredundantReceptors`
begin
Identify all the templates $t_{1,s} \notin T_S$ and write them on a file, say `file1`. Call a member of `file1` a partial receptor (initially the length of the partial receptor is just τ).
for j = 2 **to** λ-τ+1 **do**
begin
redundant ← **nil**
while not (end of `file1`) **do**
Read a partial receptor, p. If it has only left child write on `file2` the string p+'0'; if it has only right child write on `file2` the string p+'1'. If it has left and right child, create the substring tmp = `Copy`(p+'1',j,τ) and check if $tmp \in redundant$. If $tmp \notin redundant$ then add tmp to the list and write on `file2` the strings p+'0' and p+'1'. Otherwise remove from `file2` all partial receptors containing tmp and write on `file2` the string p+'1' only.
end of while.
Rename `file2` as `file1`.
end of for
end

Fig. 7 Pseudocode of the modified procedure from Fig. 2

[6] This is true if $\tau \geq \lambda/2$; when $\tau < \lambda/2$ this value must be reduced by the number of overlapping schemas induced by the templates constituting single receptor.

τ	λ=10				λ=12			
	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**
4	208	12	96	2	14	5	1231	461
5	442	28	24	0	285	18	484	5
6	672	59	5	0	1238	50	61	35
7	860	118	16	0	2471	109	23	12
8	946	245	7	0	3200	236	4	4
9	990	500	2	0	3708	492	0	0
10	1012	1012	0	0	3936	1004	0	0
11					4036	2028	0	0
12					4076	4076	0	0

Table 4. Number of rules generated by the algorithms from Fig. 2 (columns **a** and **c**) and the algorithm from Fig. 7 (column **b** and **f**). Columns **c** and **g** show the number of unrecognized strings and columns **d** and **h** - the number of holes.

4. Conclusions

In this paper we studied some ideas of generating the repertoire of receptors of minimal size. The algorithm presented in Sect. 3.3 generates receptors, each of which contains maximal number different templates. The results supported by simulations seem to be quite promising. Particularly for the set *S* consisting of 200 different strings of length λ =20, and representing a normal behaviour of a real process, the algorithm generated 42 receptors with threshold $\tau = 6$ ($P_f = 0.0334$) and 122 receptors with threshold $\tau = 7$ ($P_f = 0.0012$).

Bibliography

1. H. Bersini, F.J. Varela. Hints for adaptative problem solving gleaned from immune networks. In *Proc. of the first worκshop on Parallel Problem Soλving from Nature*, pp. 343-354, Springer: 1990.

2. P. D’haeseleer. Further efficient algorithm for generating antibody strings, Technical Report CS-95-03, The Univ. of New Mexico, Albuquerque, NM, 1995

3. P. D’haeseleer An immunological approach to change detection: Theoretical results. IEEE Computer Security Foundations Workshop, Dromquinna Manor, County Kerry, Ireland. 10-12 June 1996

4. P. D’haeseleer, S. Forrest, and P. Helman. An immunological approach to change detection: algorithms, analysis, and implications. In *Proc. of IEEE Symposium on Research in Security and Privacy*, Oakland, CA, 1996.

5. D. Dasgupta, and S. Forrest. Novelty detection in time series using ideas from immunology. In: *ISCA 5th International Conf. on Intelligent Systems*, Reno, Nevada, June 19-21, 1996 (also in [6], pp. 262-276)

6. D. Dasgupta (ed.), *Artificial Immune Systems and Their Appλications*, Springer-Verlag, Berlin Heidelberg: 1998

7. J.D. Farmer, N.H. Packard, and A.S. Perelson. The immune system, adaptation and machine-learning. *Physica D*, **22**: 187-204, 1986

8. S. Forrest, B. Javornik, R.E. Smith, and A.S. Perelson. Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation* 1(3): 191-211, 1993

9. S. Forrest, S.A. Hofmeyr, A. Somayaji, and T.A. Longstaff. A sense of self for UNIX processes. In: *Proc. of the 1996 IEEE Symposium on Research in Security and Privacy*, Oakland, CA.

10. S. Forrest, A.S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Proc. of IEEE Symposium on Research in Security and Privacy, Oakland*, CA, May 16-18, 1994, pp. 202-212

11. P. Helman, and S. Forrest. An efficient algorithm for generating random antibody strings. Technical Report CS-94-07, The University of New Mexico, Albuquerque, NM, 1994

12. J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press: Ann Arbour, 1975

13. E. Hunt, D.E. Cooke, and H. Holstein. Case memory and retrieval based on the immune system. In: M. Weloso and A. Aamodt (eds.) *Case-Based Reasoning Research and Development*, Lecture Notes in Artificial Intelligence 1010, Springer-Verlag 1995, pp. 205-216

14. J. Hunt, and D.E. Cooke. (1996) Learning using an artificial immune system. *J. of Network and Computer Applications* **19**(1996): 189-212

15. J. Hunt, *et. al*. Jisys: The development of an artificial immune system for real world applications. In [6], pp. 157-186.

16. Y. Ishida, F. Mizessyn. Learning algorithms on an immune network model: Application to sensor diagnosis. In: *Proc. Internationaλ Joint Conference on Neural Networκs*, vol. 1, pp. 33-38, China, November 3-6, 1992.

17. Ishiguru, Y. Watanabe, and Y. Uchikawa. Fault diagnosis of plant system using immune networks. In: *Proc. 1994 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems* (MFI'94), pp. 34-42, Las Vegas, October 2-5, 1994.

18. A.S. Perelson, G. Weisbuch. Immunology for physicists, *Reviews of Modern Physics*, **69**:1219-1265, 1997

19. S.T. Wierzchoń. Computer immunology: models and applications (in Polish). In: Proc. of 6-th Workshop of Polish Simulation Society, Białystok-Białowieża, 26-27 August 1999, pp. 85-87

20. S.T. Wierzchoń. Artificial immune systems. Principles and applications (in print)