An Investigation of R-Chunk Detector Generation on Higher Alphabets

Thomas Stibor[1], Kpatscha M. Bayarou[2], and Claudia Eckert[1]

[1] Department of Computer Science
Darmstadt University of Technology
{stibor,eckert}@sec.informatik.tu-darmstadt.de
[2] Fraunhofer-Institute Secure Telecooperation (SIT)
bayarou@sit.fraunhofer.de

Abstract. We propose an algorithm for generating all possible generatable r-chunk detectors, which do not cover any elements in self set S. In addition, the algorithm data structure is used to estimate the average number of generatable detectors, dependent on set size S, r-chunk length r and alphabet size Σ. We show that higher alphabets influence the number of generatable detectors in a negative manner.

1 Introduction

The biological immune system is responsible for protecting organisms against disease caused by pathogens and is able to detect and eliminate most pathogens. The immune system consists of certain types of white blood cells, called lymphocytes, that cooperate to detect pathogens and assist in the destruction of those pathogens. These lymphocytes can be thought of as detectors which recognize pathogens and destroy them, provided a binding threshold between lymphocyte and pathogen is reached. Detectors can also recognize self molecules[1], which results in autoimmune disease and can lead to death. To avoid this reaction, the immune system eliminates through a process called *negative selection*, those lymphocytes (detectors) which bind to self molecules.

From the view of computer scientists, the immune system provides a rich set of methods, ideas, principles and properties to solve computational problems [1]. A short list of immune system properties that are highly appealing from a computational perspective are :

– *Pattern recognition:* the immune system is capable of recognizing and distinguishing between self and foreign molecules.
– *Distributed detection:* the detectors used by the immune system are small, efficient, highly distributed and not subject to centralized control or coordination.
– *Anomaly detection:* the immune system can detect and react to pathogens that the body has never before encountered.

[1] produced naturally in the body

Artificial immune systems (AIS) abstract these biological methods and principles and apply them to problem oriented computational paradigms. Application of artificial immune systems are manifold (computer network security, data analysis, machine learning, search and optimization methods). In this paper we focus on anomaly detection, especially the analysis of detector generation.

2 Negative Selection Algorithm

Forrest et al. [3] developed the *negative selection algorithm* based on the *negative selection* immune system process. The algorithm operates on a representation space U, self set S and non-self set N with

$$U = S \cup N \quad and \quad S \cap N = \emptyset.$$

and returns a detector set D which recognizes elements from $U \setminus S$. The *negative selection algorithm* is summarized in the following steps.

1. Define self as a set S of elements of length l in representation space U.
2. Generate a set D of detectors, such that each fails to match any element in S.
3. Monitor S for changes by continually matching the detectors in D against S.

This principle is adaptable to nearly all computer systems, where normal system behavior (self) is appropriately mapped in self set S. A deviate from S can be recognized through the generated detectors and classified as anomaly (non-self). The problem is to generate the smallest possible detector set, which recognizes a maximum part of N. More precisely, a perfect detector set $D_{perfect}$ contains a minimal number of detectors which recognize *all* elements in $U \setminus S$. D'haesleer [4] has shown, that $D_{perfect}$ must be approximately the same size (in bits) as the self set S. Another problem arises in commonly occurring distinct self elements, which induces so called *holes*. Holes are elements from N, for which no detectors can be generated and, therefore can not be recognized and classified as non-self elements.

2.1 R-Chunk Matching

The *r-chunk* matching rule was first proposed by Balthrop et al. [5] and is an improved variant of the r-contiguous matching rule developed by Percus et al. [6]. The r-contiguous matching rule was one of the earliest rules which focused on the biological immune system as a model and abstracts the binding between antibody and antigen [6]. Informally, two elements, with the same length, match under r-contiguous rule, if at least r contiguous characters are identical. This matching rule was theoretically and practically investigated in [4,7,8]. Gonzalez et al. [8] has compared the matching performance most of the well-known (in AIS literature) matching rules. He experienced that the r-chunk matching rule achieves the highest matching performance compared with the other matching rules over the binary alphabet. Since matching performance is strongly influenced by the number of generatable detectors, we focus on the number of generatable r-chunk detectors over arbitrary alphabet sizes.

Given a space U_l^{Σ}, which contains all elements of length l over an alphabet Σ and a detector set $D \subset U_l^{\Sigma}$.

Definition 1. *An element $e \in U_l^{\Sigma}$ with $e = e_1 e_2 \ldots e_l$ and detector $d \in D$ with $d = (p, d_1 d_2 \ldots d_r)$, with $r \leq l$, $p \leq l - r + 1$* match with r-chunk rule *iff $e_i = d_i$ for $i = p, \ldots, p + r - 1$.*

Informally, element and detector match if, at position p, there is a sequence of length r where all the characters are identical.

3 Detector Generation Algorithms

We propose an algorithm called *BUILD-RCHUNK-DETECTORS* which generates all possible r-chunk detectors, which do not cover any element in S. This algorithm uses a hashtable $\mathcal{H}$ data structure to insert, delete and search efficiently for boolean values which are indexed with a key composite of r-chunk string concatenated with detector position p. Since the algorithm needs all keys from $[0..|\Sigma|^r]$ concatenated with p, $\mathcal{H}$ contains $p|\Sigma|^r$ elements, where r is the $r - chunk$ length. In addition, the hashtable is an appropriate data structure to analyse randomized operations. Figure 1 shows the hashtable with generated detectors for alphabet $\Sigma = \{0, 1\}$. The symbol | means concatenation of two elements, symbol || means logical value *true* or *false*.

```
BUILD-RCHUNK-DETECTORS($r, l, S, \Sigma$)
 1  for $i \leftarrow 0$ to $|\Sigma|^r - 1$                 /* init phase */
 2      do for $p \leftarrow 0$ to $l - r$
 3              do $\mathcal{H}.put(i|p, true)$
 4  for each $s$ in $S$                        /* label phase */
 5      do $c \leftarrow 0$
 6         while $r + c < length[s]$
 7              do $rchunk \leftarrow substring[s, r + c]$
 8                 $\mathcal{H}.put(rchunk|c, false)$
 9                 $c \leftarrow c + 1$
10  for $i \leftarrow 0$ to $|\Sigma|^r - 1$                 /* find phase */
11      do for $p \leftarrow 0$ to $l - r$
12              do $C \leftarrow \mathcal{H}.get(i|p)$
13                 if $C = true$
14                   then
15                        $D[k] \leftarrow \mathcal{H}.get(i|p)$
16                        $k \leftarrow k + 1$
17  return $D$
```

The *BUILD-RCHUNK-DETECTORS* algorithm needs four input parameters, r-chunk length r, self set S, element length l, alphabet Σ and outputs an array of all possible detectors, which do not match self elements. The algorithm is divided into three phases. The initial phase (line 1 to 3) initializes all keys with the boolean value *true*. The label phase (line 4 to 9) iterates with a length r sliding window[2] over all self elements s and replaces the hashtable boolean value, whose key matched the r-chunk with *false*. The last phase named *find* (line 10 to 17), iterates over all hashtable elements and extracts those, which have a boolean value of *true*. The returned array D contains all possible detectors which do not cover any self element.

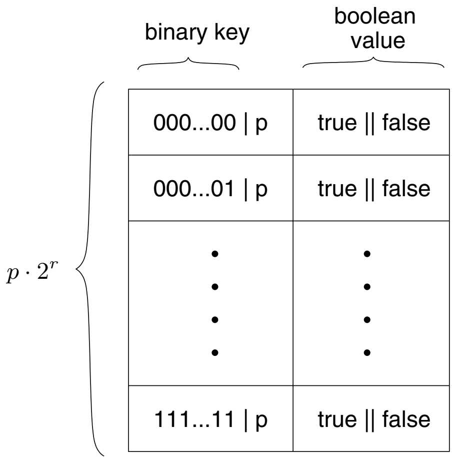


Fig. 1. Hashtable $\mathcal{H}$ configuration with $\Sigma = \{0, 1\}$

The space complexity is determinated by r and position p, where p is negligible. The hashtable uses keys of length r and, therefore the total space size results in $O(|\Sigma|^r)$. The runtime complexity is determinated by r, S and the self elements length l. The three phases need $O((l-r)\cdot|\Sigma|^r)+O(|S|\cdot(l-r+1))+O(|\Sigma|^r)$ time to generate all possible detectors. The total runtime complexity results in $O(|\Sigma|^r)$ which runs exponential in the length r. D'haesleer et al. [7] has proposed an algorithm to generate r-contiguous detectors with nearly equal runtime complexity as our proposed algorithm. He chooses r such that $|S| = O(|\Sigma|^r)$ and estimated the total runtime complexity as linear in $|S|$. This estimation is only acceptable, if $|S|$ and r are suitable chosen.

[2] substring operation

4 Detector Generation Analysis

In this section we estimate the average number of generatable r-chunk detectors. This number is determinated by the cardinality $|S|$, the element length l of S and the r-chunk length r. We use the hashtable $\mathcal{H}$ which was defined in the algorithm to estimate the average number of generatable detectors.

Proposition 1. *Given a universe U_l^{Σ} which contains all elements of length l over the alphabet Σ, r-chunks length r and a self set S randomly drawn from U_l^{Σ}, the average number of detectors which do not cover any element in S is*

$$\left(1 - \frac{1}{(l-r+1)\cdot|\Sigma|^r}\right)^{|S|\cdot(l-r+1)} \cdot (l-r+1)\cdot|\Sigma|^r$$

Proof. The hashtable $\mathcal{H}$ contains $p|\Sigma|^r$ elements. We draw $n = |S|\cdot(l-r+1)$ elements and want to find zero labeled *false* elements. The probability distribution therefore is $P(k) = \binom{n}{k} q^k \cdot (1-q)^{n-k}$. For $k = 0$ and $q = \big((l-r+1)\cdot|\Sigma|^r\big)^{-1}$ this results in

$$\left(1 - \frac{1}{(l-r+1)\cdot|\Sigma|^r}\right)^{|S|\cdot(l-r+1)}$$

and the total average number results in

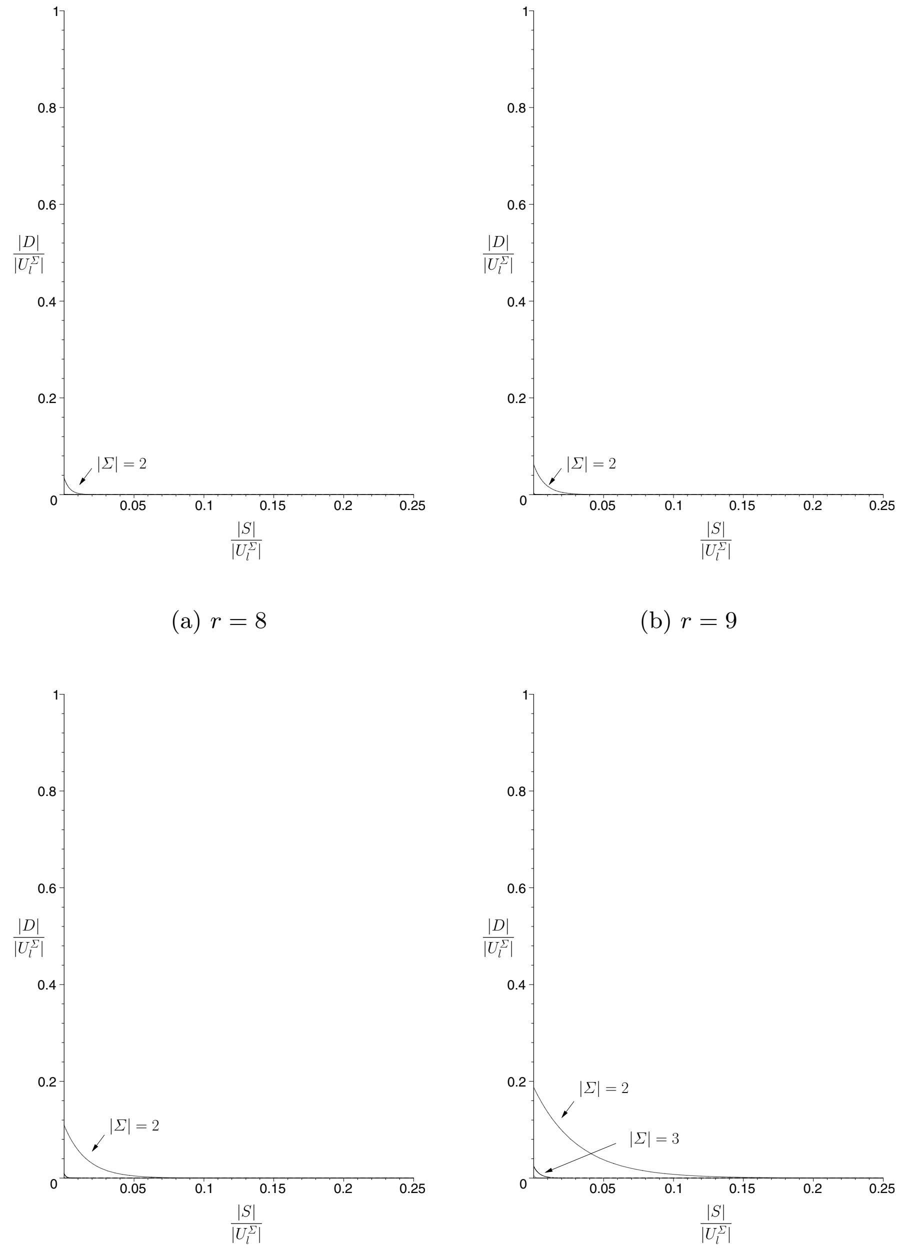
$$|D| = \left(1 - \frac{1}{(l-r+1)\cdot|\Sigma|^r}\right)^{|S|\cdot(l-r+1)} \cdot (l-r+1)\cdot|\Sigma|^r \qquad (1)$$

□

As it can be seen, term (1) strongly depends on $|S|$ and $|\Sigma|^r$. Increasing $|S|$, implies a decreasing detector set size, where $|\Sigma|^r$ also influences the amount of generatable detectors.

4.1 Number of Generatable Detectors

We investigate the parameter dependencies of $|\Sigma|, |S|, r$ and their effects on the number of generatable detectors. Therefore, we plot term (1) with small computable parameters, since term (1) increases exponentially. We choose $l = 16$, $r = 8, \ldots, l-1$, $|\Sigma| = 2, 3, 4, 5$ and select S randomly from U_l^{Σ} with a percentage proportion of $|S|/|U_l^{\Sigma}| = 0\%, \ldots, 25\%$ of total universe U_l^{Σ}. In the plots the ordinate depicts the amount of generatable detectors in proportion to the total universe. Since the universe and the amount of detectors increased with higher alphabets we represent the relative number between $|D|$ and $|U_l^{\Sigma}|$. As it can be seen in figure 2(a) to 2(c), detectors can be only generated for $|\Sigma| = 2$ and $|S|/|U_l^{\Sigma}| < 5\%$. For higher alphabets it is not possible to generate



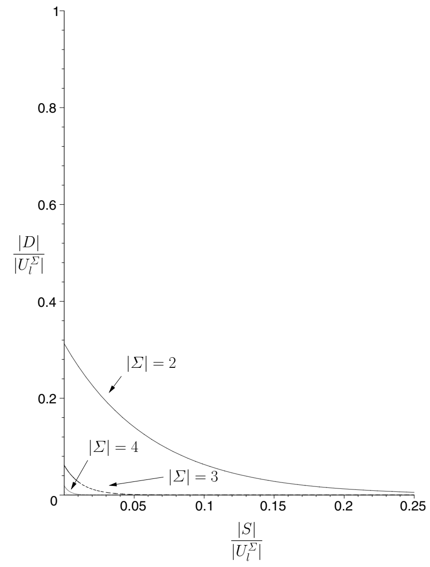
(a) $r = 8$

(b) $r = 9$

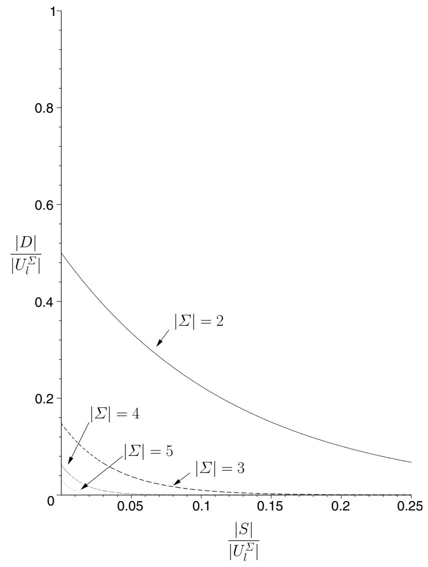
(c) $r = 10$

(d) $r = 11$

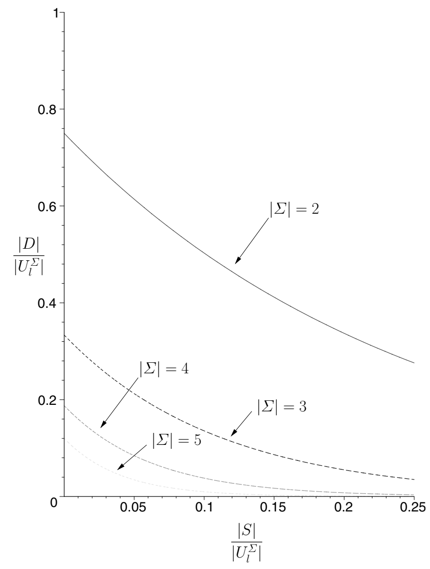
Fig. 2. (a-d) Plots of term (1), with $l = 16$, $r = \{8, 9, 10, 11\}$ and $|\Sigma| = \{2, 3, 4, 5\}$



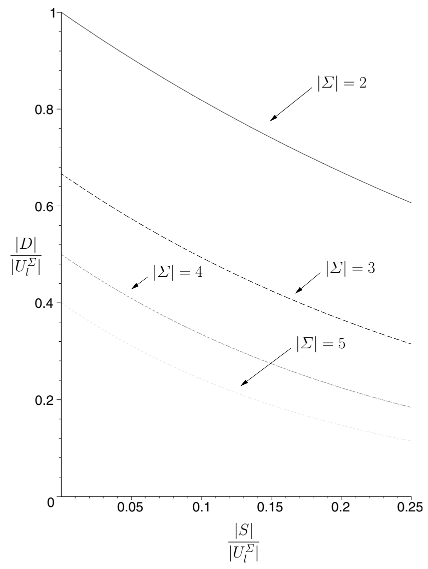
(e) $r = 12$



(f) $r = 13$



(g) $r = 14$



(h) $r = 15$

Fig. 2. (e-h) Plots of term (1), with $l = 16$, $r = \{12, 13, 14, 15\}$ and $|\Sigma| = \{2, 3, 4, 5\}$

detectors for $r \leq 10$. This phenomenon results from the r-chunk matching rule, which is not suitable for higher alphabets, since increased alphabet sizes influence the amount of generatable detectors. As it can be seen in figures 2(a) to 2(h), most detectors are generatable for an alphabet of size two. It is clear that these detectors recognize less elements from U_l^{Σ} than higher alphabets detectors for same value of l and r. But, as we see in figures 2(a) to 2(h), increasing the alphabet size implies less generatable detectors for a fixed r. To generate a destined amount of detectors for arbitrary alphabet sizes, the r-chunk length must lie near l, which results in larger space complexity. If $|\Sigma| > 5$ and $r > 16$ it is not feasible to generate all possible detectors, due to the large space complexity.

5 Conclusion and Future Work

We have proposed and analyzed an algorithm which generates all possible r-chunk detectors over arbitrary alphabet sizes. In addition, an average analysis was shown, to estimate the number of generatable detectors, by given parameters l, r, S and Σ. It has been shown, that the alphabet size has a strong influence on the number of generatable detectors. For the r-chunk matching rule, the alphabet size two achieves the highest number of generatable detectors. Higher alphabets bias the amount of generatable detectors in proportion to total universe. To generate a sufficiently large number of detectors r must lie near l and this results in an infeasible space complexity. So far, the total number of generatable detectors was not considered in terms of non-self cover. The further work will be to investigate the total non-self cover on arbitrary alphabet sizes.

Acknowledgments. The authors are grateful to Bob Lindsey for his comments and annotations.

References

1. Leandro N. de Castro, Jonathan Timmis: Artificial Immune Systems: A New Computational Intelligence Approach. Springer-Verlag (2002)
2. Anil Somayaji, Steven Hofmeyr, Stephanie Forrest: Principles of a computer immune system. In: Meeting on New Security Paradigms, 23-26 Sept. 1997, Langdale, UK, (New York, NY, USA : ACM, 1998) 75–82
3. Forrest, S., Perelson, A.S., Allen, L., Cherukuri, R.: Self-nonself discrimination in a computer. In: Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, Oakland, CA, IEEE Computer Society Press (1994) 202–212
4. D'haeseleer, P.: An immunological approach to change detection: Theoretical results. In: Proceedings of the 9th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press (1996)
5. Balthrop, J., Esponda, F., Forrest, S., Glickman, M.: Coverage and generalization in an artificial immune system. In: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, Morgan Kaufmann Publishers (2002) 3–10

6. Jerome K. Percus, Ora E. Percus, Alan S. Perelson: Predicting the Size of the T-Cell Receptor and Antibody Combining Region from Consideration of Efficient Self-Nonself Discrimination. Proceedings of National Academy of Sciences USA **90** (1993) 1691–1695
7. P. D'haeseleer, S. Forrest, P. Helman: An immunological approach to change detection: algorithms, analysis, and implications, Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy (1996)
8. F. Gonzalez, D. Dasgupta, J. Gomez: The effect of binary matching rules in negative selection. In: Genetic and Evolutionary Computation Conference. (2003)