J Math Model Algor (2007) 6:411–431
DOI 10.1007/s10852-007-9061-x

Coevolutionary-based Mechanisms for Network Anomaly Detection

Marek Ostaszewski · Franciszek Seredynski · Pascal Bouvry

Received: 1 November 2005 / Accepted: 1 December 2006 /
Published online: 22 March 2007

Abstract The paper presents an approach based on the principles of immune systems applied to the anomaly detection problem. Flexibility and efficiency of the anomaly detection system are achieved by building a model of the network behavior based on the self–nonself space paradigm. Covering both self and nonself spaces by hyperrectangular structures is proposed. The structures corresponding to self-space are built using a training set from this space. The hyperrectangular detectors covering nonself space are created using a niching genetic algorithm. A coevolutionary algorithm is proposed to enhance this process. The results of experiments show a high quality of intrusion detection, which outperform the quality of the recently proposed approach based on a hypersphere representation of the self-space.

Key words artificial immune systems · anomaly detection problem · self–nonself space paradigm · hyperrectangular detectors · coevolutionary algorithms · computer networks

Mathematics Subject Classifications (2000) 90B18 · 90B20 · 90C99 · 68T20 · 68U20

M. Ostaszewski · P. Bouvry
Faculty of Sciences, Technology and Communication, University of Luxembourg,
6 rue Coudenhove Kalergi, 1359 Luxembourg-Kirchberg, Luxembourg

F. Seredynski (✉)
Polish–Japanese Institute of Information Technology, Koszykowa 86,
02-008 Warsaw, Poland
e-mail: sered@ipipan.waw.pl

F. Seredynski
Institute of Computer Science, Polish Academy of Sciences, Ordona 21,
01-237 Warsaw, Poland

1 Introduction

Security has become an extremely important topic in recent years, especially when referring to computer systems connected to the Internet. Detection of intrusions in computer networks turned out to be difficult task, mostly because the continuous evolution of methods and tools used for unauthorized access to various network resources. Many approaches have been applied to problem of the recognition of malicious activities appearing in network traffic. Traditional methods are based on the recognition of attack signatures [16], however the list of signatures must be constantly updated to recognize new attacks. Additionally this method prevents only against attacks that are on the signature list. Other methods of anomaly detection like statistical model of the system behavior [6] and data mining analysis of network traffic [11] have also been a subject of a research. However these approaches introduce certain problems related to the computational complexity of analyzing large amounts of data. Anomaly detection was also analyzed by methods based on visualization [1]. Algorithms inspired by Nature are also being applied to domain of network traffic anomaly detection: in particular, paradigms like genetic algorithms and particle swarm search [5] or combining evolutionary algorithms with fuzzy set theory [12].

An artificial immune system (AIS) is a computational paradigm based on abstracting natural immunological processes [9], which can be applied to solve problems of computer security, including the detection of intrusions and anomalies [10, 19]. AIS offers alternative mechanisms to deal with unwanted activities in computer networks, including generalization and recognition of previously unknown attacks. A recent promising approach to network anomaly detection has been presented by [4], and is based on a description of legitimate behavior using hypersphere structures. We propose to use hyperrectangle structures to provide a more precise definition of the normal traffic and coevolutionary-based mechanisms to enhance the process of anomaly detection.

The paper is organized as follows: the coming section contains a short definition of self–nonself paradigm, originated from AIS and its application to the network anomaly detection. The principles of construction of an effective model of network behavior called self space are presented in Section 3. Nonself space detectors development, using niching genetic algorithm and coevolutionary techniques are described in Section 4. Section 5 contains Nonself detection process using model of abnormal network traffic. The results of the performed experiments are presented in Section 6. Last section contains the conclusion and a discussion of further possibilities of development of the presented approach.

2 Immune Anomaly Detection

Self–nonself paradigm offered by AIS methodology seems to be a proper guideline for constructing a network behavior classification system. The self space corresponds to the organism protected by its natural immune system, which consequently cannot attack or define as enemy any defended cells. In the network domain, the self space is defined on the basis of normal traffic and the nonself space contains all possible threats and deviations. They are complementary, the same way it is for an organism

and its environment in Nature. Self–nonself space principles are coupled with a negative selection algorithm [8], which is used to construct detectors focused on nonself space. An accurate model of a space is required to effectively deal with an anomaly detection. Stibor et al. [18] indicate that the negative selection over Hamming shape-space suffers from several weaknesses. In effect the model of the space presented in [4] has been taken under consideration. Although [17] indicates that real-valued negative selection may cause severe problems for system scalability and detector set generation, a similar approach presented in this paper seems to be worth of taking under consideration.

Main steps of the anomaly detection proposed in this paper are similar to negative the selection algorithm proposed by [8], and go as follows:

- construct *Self* structures using normal network traffic
- construct *Nonself* detectors using obtained *Self*
- perform detection over network traffic using *Nonself* detectors.

The differences from actual negative selection lay in the description of the normal activity and, in effect, in the construction of the *Nonself* detectors. Our work tends to represent *Self* and *Nonself* as complementary, multidimensional subspaces of space of all activities rather than a collection of strings. As it will be presented in following sections, the process of matching of detector and examined structure is also slightly different than the process of matching of strings.

3 Self Space Construction

Self space is constructed using the recorded parameters of the normal traffic that describe the desired features of a model of regular behavior. Assuming that certain number k of parameters of the network traffic will be measured in a defined time interval, k-tuples of time series can also be defined as a set of points in k-dimensional space, where every parameter from a set of measured parameters is a separate dimension. Every one of these points describes a certain **state** of the monitored network in a **state space** of all possible combinations of the chosen parameters (after [4]).

Recording all regular values of network traffic is not enough to build proper model of normal behavior. Additionally, it is required to enhance the model by a generalization ability. This can be achieved by describing incoming values that are recognizable as similar to *Self*, and not only the exact same than those already recorded. The acceptable values are varying from recorded ones in a certain range, i.e. a level of deviation from given data that can be defined with a parameter, henceforth called variability parameter $\boldsymbol{v}$. It was used by [4] as a parameter of spherical shape for *Self*. In their design $\boldsymbol{v}$ is the radius of a hypersphere centered in the recorded state, and hyperrectangular *Nonself* detectors (design and development of detectors is covered in following section). A spherical construction of *Self* has a drawback when it comes to the detector construction, because the generated detectors have no chance to cover the Nonself space completely. Figure 1a illustrates this case. As one may see, there are subspaces that cannot be covered by the detectors due to the shape conflict between the structures used for *Self* and *Nonself* description. Additionally, hyperspherical construction of *Self* cannot distinguish

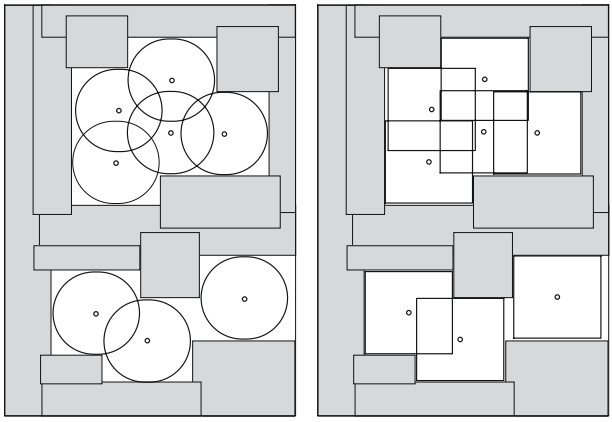

Fig. 1 Self space and detector sets for **a** spherical **b** rectangular construction of *Self*

between the different dimensions providing only one value (hyperradius) for every dimension. Figure 1b presents our approach to construct a hyperdimensional space, where a dimension (in this case 2D) depends on the number of parameters used for monitoring process. The parameter $\boldsymbol{v}$ describes the similarity degree by defining intervals of similarity for every recorded state. These intervals may have different values for every dimension taken into consideration and allow to construct more accurate *Self*. Henceforth, $\boldsymbol{v}$ for k parameters (dimensions) is defined as a vector $\boldsymbol{v} = (\boldsymbol{v}^1, \ldots, \boldsymbol{v}^k)$, with every value of variability $\boldsymbol{v}^i$ responding to certain, monitored parameter.

Main steps of the process of *Self* development can be enclosed in the following algorithm (henceforth referred as Self Development Algorithm):

Self Development Algorithm:
1: Select *Initial data* = TCP data collected in real system (*tcpdump* format)
2: Select set of monitored traffic parameters $P = \{P_1 \ldots P_k\}$
3: Select time interval Δt for counting parameters from P
4: Calculate time series R using *Initial data, P* and Δt
5: Normalize R to [0.0,1.0] interval
6: Select value w of sliding window for R transformation
6a: if $w>1$ **then** transform R into W
6b: else W = R
7: Calculate training set TR and testing set TS from W
8: Calculate variability parameter $\boldsymbol{v}$ using TR
9: Select scale factor *sf* for $\boldsymbol{v}$
10: Construct *Self* using TR, *sf* and $\boldsymbol{v}$
11: Adjust *Self* using TS, *sf* and $\boldsymbol{v}$.

Parameters of set P in step **2** are variables used for description of network traffic, and their values are measured and counted every Δt, creating in step **4** k-variate time series of n entries (tuples) $R = r_1, r_2, \ldots, r_n$ where $r_i = (r_i^{P_1}, r_i^{P_2}, \ldots, r_i^{P_k})$. As it was mentioned above, every r_i is treated as a point in k-dimensional state space, and will be also referred as a state. Time series R is then normalized to [0.0,1.0] interval, taking into account every of parameters from P.

A sliding window transformation of R into W in step **6a** is performed using parameter w to achieve temporary aggregations among values of time series. Transformed time series W that can be also described as windowed time series is created from R in following way:
$W = \{(r_1, \ldots, r_w), (r_2, \ldots, r_{w+1}), \ \ldots\ , (r_{n-w+1}, \ldots, r_n)\}$, what can be written as $W = \{rw_1, \ \ldots, \ rw_m\}$, where $m = n - w + 1$ and $rw_i = \frac{r_i + r_{i+1} + \cdots + r_{i+w}}{w}$. Taking into account, that state r_i consists of k elements, sliding window transformation is performed for every of them independently, giving $rw_i = (rw_i^{P_1}, \ldots, rw_i^{P_k})$. For sliding window size $w = 1$ every recorded state remains unchanged and $r_i = rw_i$ therefore windowed time series W are identical to time series R, what is illustrated in step **6b**.

It is necessary to emphasize, that a sliding window transformation preserves the k-dimensional structure of the state space, but modifies a number and values of time series tuples – states. Time series of recorded three parameters, i.e. $R_1 = \{(r_i^{P_1}, r_i^{P_2}, r_i^{P_3}), (r_{i+1}^{P_1}, r_{i+1}^{P_2}, r_{i+1}^{P_3}), (r_{i+2}^{P_1}, r_{i+2}^{P_2}, r_{i+2}^{P_3})\}$, and a sliding window of a size equal to 3 will create transformed time series $Rw_1 = \{(r_i^{P_1} + r_{i+1}^{P_1} + r_{i+2}^{P_1})/3, (r_i^{P_2} + r_{i+1}^{P_2} + r_{i+2}^{P_2})/3, (r_i^{P_3} + r_{i+1}^{P_3} + r_{i+2}^{P_3})/3\}$, giving one averaged tuple instead of three. Dasgupta and Gonzalez [4] apply different kind of sliding window fashion to achieve aggregations between parameters. A number of P elements is increased by treating succeeding values of monitored parameters in sliding window as additional parameters. Data from previous example will create a 9-dimensional tuple, $R_1 = \{r_i^{P_1}, r_i^{P_2}, r_i^{P_3}, r_i^{P_4}, r_i^{P_5}, r_i^{P_6}, r_i^{P_7}, r_i^{P_8}, r_i^{P_9}\}$, where $r_i^{P_1}, r_i^{P_2}, r_i^{P_3}$ are $r_i^{P_1}, r_{i+1}^{P_1}, r_{i+2}^{P_1}$, $r_i^{P_4}, r_i^{P_5}, r_i^{P_6}$ are $r_i^{P_2}, r_{i+1}^{P_2}, r_{i+2}^{P_2}$ and $r_i^{P_7}, r_i^{P_8}, r_i^{P_9}$ are $r_i^{P_3}$, $r_{i+1}^{P_3}$, $r_{i+2}^{P_3}$. This transformation also gives one tuple, but with a different structure, and monitoring n parameters with a window size w creates $n * w-$dimensional state space and greatly increases computational complexity.

The construction of the *Self* space is based only on a normal traffic data, which has to be divided into two subsets, training and testing. The first one is used for constructing a model, the second one for validating it. Assigning W elements to TR and TS stated in step **7** is performed randomly according to the defined proportions. Having TR and TS, the next step of *Self* space development is the computation of $\boldsymbol{v}$. In contrary to authors in [4] this approach takes into account calculation of $\boldsymbol{v}$ on the basis of normal traffic data.

In step **8** elements of $\boldsymbol{v}$ are calculated, using states from TR. First, for every state form TR a set of states being its closest neighbors (`ClosestSet`) is built. Then for every parameter from P a minimal distance is found, and such a vector of minimal values is then added to set of minimal distances `MinSet`. `MinSet` is then used for calculation of standard deviation for every parameter from P giving in effect the variability vector $\boldsymbol{v}$. Algorithm of $\boldsymbol{v}$ construction goes as follows:

Variability Calculation Algorithm:
minDist[`PSize`], v[`PSize`], MinSet = *null*, ClosestSet = *null*
for each State → st **from** TR
  cSet = set of States closest to st from TR\{st}
  ClosestSet.*add*(cSet)
  for i = 1 to `PSetSize`
    for each State → clst **from** ClosestSet
      `minVal` = st.*valueOfParameter*(i) – clst.*valueOfParameter*(i)
      if `minFinal` > `minVal` **then** `minFinal` = `minVal`

```
      minDist[i] = minFinal
    MinSet.add(minDists)
for i = 1 to numOfParams
  for each minDist → tab from MinSet
      stDev =calculateStandardDeviation(tab[i])
    v[i] = stDev.
```

An additional parameter, scaling factor *sf* is needed to provide a control over *Self* development. It was stated, that $\boldsymbol{v}$ describes the similarity of incoming patterns of network behavior to those already recorded. Therefore, $\boldsymbol{v}$ has a clear impact on the tolerance level of future detection process, allowing or disallowing certain patterns to be considered as normal. The parameter *sf*, defined in step **9** is a coefficient that scales $\boldsymbol{v}$ and provides more, or less tolerant ***Self***.

Having $\boldsymbol{v}$ scaled with *sf* parameter, step **10** describes initial *Self* construction. For every state from TR set a structure is created consisting of two vectors, *low* and *high* of *P* size, where a pair low^j and $high^j$ describes an interval for the parameter *j* considered as normal. These two vectors are constructed using elements from TR, rw_i and **v**, that $low = (rw_i^{P_1} - \boldsymbol{v}^1, \ldots, rw_i^{P_n} - \boldsymbol{v}^n)$ and $high = (rw_i^{P_1} + \boldsymbol{v}^1, \ldots, rw_i^{P_n} + \boldsymbol{v}^n)$. Structure consisting of these two vectors is then added to *Self* set.

Finally, in step **11**, *Self* space is adjusted using previously created test set TS. Every element from TS is being checked, if it is covered by *Self* structures created on the basis of TR, i.e. if it fits all *low* and *high* intervals of any structure from *Self*. If it is not the case, a new structure is created, for an element from testing set by applying $\boldsymbol{v}$, as it was done for structures from TR.

4 Nonself Space Construction

Nonself space is complementary to *Self*, therefore it should contain all unwanted patterns of the network traffic, i.e. values of incoming time series that will be anomalous. As it was presented in the previous section, *Self* consists of interval vectors, built for every state from training set and some states from testing set. *Self* development effects with numerous hyperrectangles of the same size (couple of thousands, or even more – see, Section 6.1). It would be desirable, to cover the *Nonself* space with few detectors of different sizes, most likely covering more space than *Self* structures (see Fig. 1). This means, that a *Nonself* detector will also be hyperrectangular structure based on *low* and *high* vectors, but with a more flexible construction, without using an unique parameter, in contrary to using $\boldsymbol{v}$ for *Self*. Dasgupta and González [4] proposed a genetic algorithm with sequential niching (NGA) for this purpose, that is solving a multimodal and multiobjective problem, what is also applied in our work.

Having *Self* developed using a certain scale of $\boldsymbol{v}$, a subset of *Nonself* detectors is constructed as a complementary space to *Self*. It is also possible to develop additional detector sets, with different scale factor values, obtaining various security models. Figure 2a illustrates *Self* for a relatively small value of *sf* and, in effect, $\boldsymbol{v}$, which implies that the detectors of *Nonself* have a greater possibility of raising a false alarm (false positive error), but less risk of leaving any anomaly undetected (false negative error) [7]. On the other hand, Fig. 2b, shows a higher false negative ratio, with a lower number of false positives. Figure 2c presents a *Self* constructed

Fig. 2 Self space and detector sets for **a** low and **b** high value of $\boldsymbol{v}$, and **c** detection levels in summary

for two combined *sf* values, and shows, that the detector sets are different, like in the *Nonself* classification process. One can see that scaled $\boldsymbol{v}$ used for the construction of *Self* influences the tolerance, and allows to construct complex security model. Therefore the process of *Nonself* space detectors construction can be described by the following algorithm (henceforth referred as Nonself Development Algorithm):

Nonself Development Algorithm:
1: Select set of security levels $SecLevels = \{sf_1 \ \ldots \ sf_m\}$
– set of scaling factors in ascending order
2: Select set of detector levels $DetLevels = \{Nonself_1 \ \ldots \ Nonself_m\}$
corresponding to security levels – *SecLevels* elements
3: Select parameters of NGA for detector set construction
3a: if (coevolutionary NGA) **then**
select coevolutionary population set *CoPopulation*
select coevolution parameters
4: Calculate $\boldsymbol{v}$ for normal traffic
5: for i=1 **to** m **do**
5a: Construct *Self* using $\boldsymbol{v} * sf_i$ from *SecLevels*
5b: Construct *Nonself* using NGA applied to *Self*
5c: $DetLevels_i = Nonself$.

Step **1** of the algorithm is definition of security levels, and immediately after, in step **2** there is set of detector sets (initially empty), corresponding to the *SecLevel* values. The security levels have to be sorted out by a growing value of $\boldsymbol{v}$ that were used to construct *Self*. The highest level defines the set most tolerant to abnormalities.

4.1 Niching Genetic Algorithm

Step **3** is the definition of the parameters that control the work of NGA used for the *Nonself* construction. Creating a detector set is a complex multiobjective problem.

The goal is to cover the space as large as possible with detectors that avoid *Self* elements and cover an unique space, without overlaying the same part *Nonself*. A population of NGA consists of individuals constructed from a pair of vectors, reflecting *high* and *low* values of a conditional part of a detector. Therefore, a real-coded NGA [13] has to be applied to this problem, as long as vectors of real values ([0.0, 1.0] interval) are used. NGA has to deal with a multiobjective problem and for this reason sequential niching method [2] has been applied to cover the different subspaces of *Nonself* with volumes as large as possible. During each run of NGA the fitness function is modified in order to focus the process of searching for new detectors in the subspaces that are not covered by the previously evolved detectors. NGA schema, using a set of *Self* structures, goes as follows:

Niching Genetic Algorithm (NGA):
Self, Nonself =*null*, `numAtt`=0, `numDet`=0
while `numDet` < `maxDet` **and** `numAtt` < `maxAtt`
  runGA(Self)
  D ← best evolved detector
  `fit` = *calculateFitness*(D)
  if `fit` > `minFit`
    Nonself.*addDetector*(D)
    `numAtt` = 0
  else
    `numAtt` = `numAtt` + 1
end while
return Nonself.

The parameter `minFit` is the minimal value of fitness expected from evolved detector, `maxAtt` and `numAtt` are, respectively, the maximum and the current number of attempts to evolve a single detector and `maxDet` and `numDet` are, respectively, the maximum and the current number of detectors in the detector set. Details on selection of NGA parameters are presented in Section 6.1.

The fitness is computed by taking under consideration three factors, as shown below:

Volume calculation – a volume of a given detector is calculated as follows:

$$Volume(D) = \prod_{i=1}^{n} (high^i - low^i),$$

where *high* and *low* are elements of the vectors on the positions corresponding to the used parameters.

Overlaying with Self structures – a volume of the space overlayed with *Self* is computed as follows:

$$SelfOverlay(D) = \sum_{i=1}^{m} (D \cap xs_i),$$

where xs_i is a *Self* structure.

Overlaying with already developed detectors – a volume of the space overlayed by already developed detectors is calculated as follows:

$$DetectorOverlay(D) = \sum_{i=1}^{k} (D \cap D_k).$$

The fitness of a single detector is calculated from the equation

$$Fitness(D) = Volume(D) - (SelfOverlay(D) + DetectorOverlay(D)).$$

4.2 Coevolutionary Mechanism in the Detector Generation Process

Defined in step **3a** of Nonself Development Algorithm coevolutionary NGA is our proposition for enhancing *Nonself* detector generation process using coevolutionary mechanisms. Although NGA tends to cover all *Nonself* in the most efficient way, this process is unsupervised and the goal for created detectors is based only on constraints. The criterion of volume does not specify, which exact subspace of search space detector should cover, therefore a goal is not precisely defined. Additional information could lower the computational cost of the generation process and, in effect, give more specialized detectors covering certain subspaces in *Nonself*. For that purpose a coevolutionary algorithm is used. Coevolution is relatively new research paradigm in the field of evolutionary computation. The basic idea is taken from the world of Nature, where two or more coexisting species are constraining one other to evolve better features. Among many coevolution models, one seems to be useful to the detector generation problem. *Predator–prey* paradigm [15] describes a model, where individuals of one type (predators) are trying to catch individuals of another type (preys). The population of the first species develops features that allow it to catch its prey easily, and attributes of the second one evolve to make escape from a predator possible. Applying this process to the detector generation mechanism could improve it, by providing a certain goal in search space. Coevolution allows controlling the process, by enforcing on generated detectors certain features, indicating areas to cover. Some definitions [15] have to be assumed to apply coevolutionary algorithms to the detector generation problem:

Constraint Satisfaction Problem (CSP): a class of problems effectively solvable by coevolutionary algorithms. The first of two coevolving populations is a population of solutions (henceforth called *Solutions*), and the other one is a population of constraints (henceforth called *Constraints*) that *Solutions* have to fit. Because of their static nature, *Constraints* cannot evolve but their fitness can be also evaluated.
Encounter: a confrontation between individuals from *Solutions* and *Constraints* results in the a victory of one and the loss of the other. A *Solution* wins if it fits given *Constraint*, and loses if *Constraint* cannot be satisfied by a given solution.
LifeTime Fitness Evaluation (LTFE): in opposite to the classic GA, every individual is tested multiple times and has a list of his encounters that changes, as it might be said, through its lifetime. A fitness is calculated on the basis of confrontations with individuals from coevolving population. LTFE regulates a number of confrontations, and thus, affects calculated fitness. Probability of choice to encounter depends on fitness, therefore, even if *Constraints* cannot evolve, winning ones are tested more frequently against *Solutions*.

To apply the coevolutionary algorithm to the detector generation problem, the second population has to be assumed, named *CoPopulation* in step **3a**. To define the proper constraints for the detectors, a set of anomalies has to be constructed. Similarly to the *predator–prey*, the proposed model considers a situation when individuals from the detector set (*Solutions*) try to intercept individuals from a set of anomalies (*Constraints*). An anomaly (individual) is defined as a certain state from *Nonself* space in the form of a vector $a = (a_1, ..., a_n)$, where a_i is a value for corresponding parameter. An encounter between a detector and an anomaly leads to an assessment checking if the anomaly is placed inside the subspace covered by the detector. The detector wins the encounter if it intercepts the given anomaly, otherwise the constraining state is the winner. The proposed coevolutionary algorithm can be defined as follows:

for each Detector → det **from** DetectorSet
  for i=1 to LTFE
    anom = AnomalySet.*chooseAccordingToFitness*()
    result = *encounter*(det, anom)
    det.*updateHistory*(result)
    anom.*updateHistory*(result)
  AnomalySet.*updateFitness*().

Although anomalies are constant, they have their own fitness based on results of encounters. They are chosen to further encounters on the basis of their fitness, so anomalies that avoid detectors more efficiently are chosen and tested more often. A fitness of a given detector is finally computed after running the coevolutionary algorithm as follows:

$$Final\,Fitness(D) = (1 + Encounter\,History(D)) * Fitness(D),$$

where $Encounter\,History(D)$ is a function returning the summarized effect of all encounters for a given detector.

This way, the generated individuals are focused on certain subspaces, and specialized in capturing states from *Nonself* defined as the coevolving population. This process is similar to vaccination: the way the specialization of antibodies in the human immune system is achieved by presenting negative examples. The details on the selection of coevolutionary parameters and *Co Population* design are covered in Section 6.4.

5 Detection Process

Having m detector sets, one set for each security level, constructed using the Nonself Development Algorithm, the *Nonself* detection process will be performed by checking every state of the monitored time series and returning the proper level of danger caused by them, as follows: $analyse(state_i) = max(\{num(Det\,Levels)\} \cup \{0\})$, where *num(DetLevels)* returns the index of detector set that raises an alarm. The detector set of certain security level l can be defined as follows: $Det\,Levels_l = \{D_1, \ldots, D_n\}$, where D_j : *if* $state_i$ *is intercepted then Nonself*, and $intercepted(state_i)$ returns `true` or `false` result of following formula: $state_i^{P_1} \in [low_j^1, high_j^1] \wedge \ldots \wedge$

$state_i^{P_k} \in [low_j^k, high_j^k]$. A single detector classifies if a given state is captured by the covered space defined in the conditional part by the two vectors *high* and *low*. The construction of a space from the junction of intervals in every dimension is similar to the case of *Self* elements.

Therefore, incoming network traffic is monitored and modified according to parameters used in Self Development Algorithm, i.e. using the same P, Δt, w and normalization parameters. Then, transformed values are compared to see, if any of detectors at any level will intercept this state.

Additional parameter, that describes anomaly detection process is an interval of alarm counting $\Delta ta = c * \Delta t$ and $c = 1, 2...$. It describes a number of time intervals Δt over which results will be summed, and presented. If the coefficient c is equal to 1, alerts will be presented on the fly, according to time series interval. If $c > 1$ then a certain delay is introduced, after which the sum of the alerts is presented.

6 Experimental Results

A number of experiments have been performed to find out the effectiveness of self–nonself approach to the anomaly detection problem, based on hyperrectangular *Self* structures and involving the coevolutionary algorithm. The *Self* space for this experiment was constructed using real-world data [14] and according to the steps of the Self Development Algorithm introduced in Section 3. Step **1** data was the first week of the collected outside network traffic using *tcpdump*, which was unaffected by anomalies, and for one of chosen computers figuring IP 172.16.114.50 (marx). The set P in step **2** was constructed of the number of bytes per second (P1), the number of packets per second (P2) and the number of ICMP packets per second (P3). Time interval in step **3**, Δt, was equal to 1 second. In effect time series R was constructed, with states of the system taking form of the triplet $r_i = \{r_i^{P_1}, r_i^{P_2}, r_i^{P_3}\}$ as a point in the 3-dimensional space. The *Self* structures created from the first week data in a proportion 70:30 of training (TR) to testing (TS) sets (step **7**). Further experiments involve different *sf* and and *w* values, but if it is not explicitly mentioned, *w* and *sf* are by default equal to 1. A number of *Self* structures for this default configuration was about 5,800.

6.1 Experiment #1 – Generation of the Detector Set for Different *sf* Values

In the first experiment GA was used for the development of self space and detector sets for different values of *sf*. NGA was run with the maximum number of runs equal to 20, the maximum number of attempts to evolve a rule is equal to 15, the number of generations is equal to 750 and a population size equal to 100. The following GA operators have been used [13]: a tournament selection with the tournament size equal to 2, vector crossover, Gaussian mutation with probability 0.1 and border mutation with border values 0 and 1, and probability 0.01.

Figure 3 shows the comparison between the covered *Nonself* space with the detectors developed for the case of two values of *sf*. Figures 3a and 3b show only a part of the generated detectors. The size of the detector set in both cases is equal to 15. Incase of small (0.3) value of *sf* the number of Self structures was greater, about 5,900.

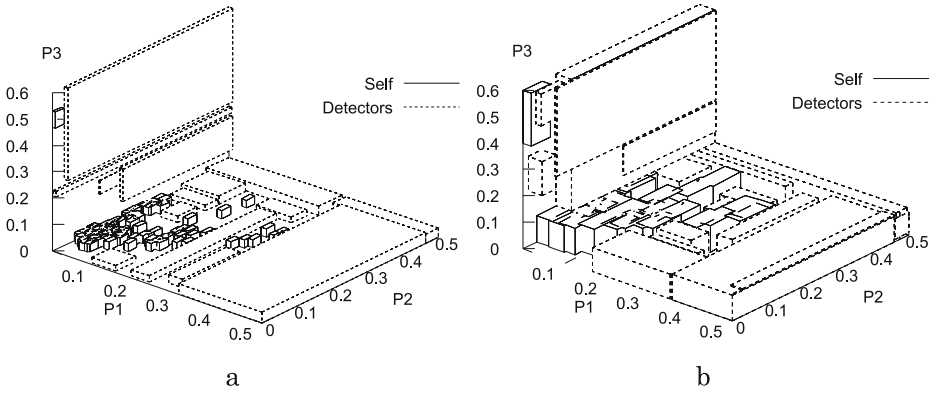

Fig. 3 Self space and detector sets for *sf* equal to **a** 0.3 and **b** 1.0

6.2 Experiment #2 – Anomaly Detection Process for Different *sf* and *w* Values

Detector sets generated by NGA have been used to anomaly detection process on the second week of mentioned, MIT data. To avoid gaps that could be observed on Fig. 3 greater maximum number of detectors was introduced, namely equal to 50. This number is over 100 times smaller than *Self* size.

The second week contains five simulated attacks, one for every day of the network traffic, as shown in Table 1.

The parameter Δta was equal to $60 * \Delta t$, so alerts generated by detectors are gathered over 1 min time interval. The results of monitoring the anomaly detection process are presented in Figs. 4 and 5. Figure 4a presents anomalies detected by the set of detectors for $sf = 0.3$ and $w = 1$, and Fig. 4b for $w = 3$. Figure 5 presents detection effects for the set with $sf = 1.0$ and $w = 1$ (Fig. 5a), and $w = 3$ (Fig. 5b). The analysis of these figures indicates greater sensitivity of detectors constructed for Self with $w = 3$, what can be explained, if temporal patterns are taken under consideration. With a larger window size, one can intercept time dependencies between preceding and succeeding states, what is impossible for the detectors based on $w = 1$.

Figure 5 indicates that monitoring using both detector sets would result in discovering all five attacks, though having relatively high level of *sf*. As one may notice, some peaks in these figures (1,485 and 4,491 min in Fig. 4a, 4,498 in Fig. 5a and 1,487 in Fig. 5b) are groups of multiple lines. After the analysis of Table 1 it is possible to notice, that the duration of attacks 2 and 4 was relatively long and that the system

Table 1 MIT second week attacks

Day	Name	Type	Start	Duration
1	Back	DoS	9:39:16	00:59
2	Portsweep	Probe	8:44:17	26:56
3	Satan	Probe	12:02:13	02:29
4	Portsweep	Probe	10:50:11	17:29
5	Neptune	DoS	11:20:15	04:00

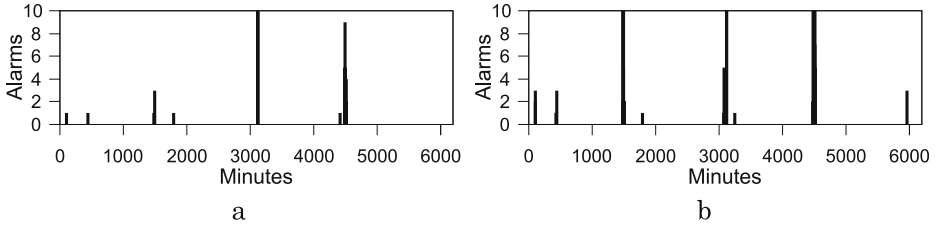

Fig. 4 Attacks detected using $sf = 0.3$ and w equal to **a** one and **b** three

raised more than one alarm during the monitoring process. Those attacks on Figs. 4b and 5b are indicated as groups of lines having an alarm number equal to or larger than 10, what makes them look like bold lines. An interesting fact can be observed after the comparison of alarms raised for each attack – obviously, the probe attacks are recognized with a greater accuracy than DoS attacks. Additionally, the window size seems to have optimal values for every attacks, as it may be observed in Fig. 5, where $w = 1$ manages to capture the first attack, but misses the last one, and for $w = 3$ the first attack remains unreported, but last one is displayed. An interesting case is the attack number three, indicated with a great strength in every parameter configuration though relatively short duration time. It may be explained by dependencies between an attack type and the structure and parameters used in *Self* construction.

More experiments were performed, concerning larger w parameter and more sf values were carried out. In effect ROC (Receiver Operating Characteristics) [7] diagram has been constructed. It is presented on Fig. 6 and shows the classification performance of the detector sets for a given window size equal to 1, 3, 5 and 7, respectively. Points marked on each curve correspond to sf values equal to 1.3, 1.0, 0.7 and 0.3, respectively. One can notice that for a given window size the detection rate grows, when value of sf decreases. It is worth noticing that the window size influences the precision of detection, and that the detector set constructed for $w = 7$ performs well even for relatively high sf, what results in decreasing the number of false alarms.

6.3 Experiment #3 – Anomaly Detection Process for Spherical Construction of *Self*

The approach presented by [4] is based on the detector sets developed for *Self* shaped as hyperspheres, which are created using a given state of the system as a cen-

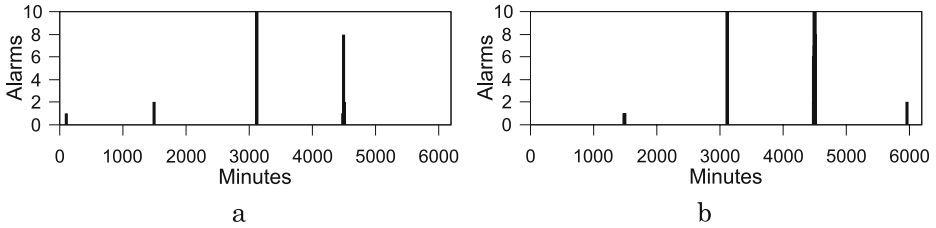

Fig. 5 Attacks detected using $sf = 1.0$ and w equal to **a** one and **b** three

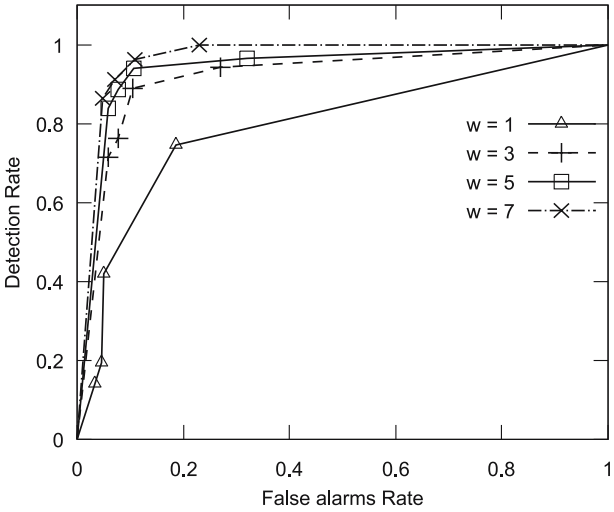

Fig. 6 ROC diagram for different w sizes

ter and single value of $\boldsymbol{v}$ as a hyperradius. Experimental results presented by [4] show that a hyperspherical design of *Self* was sufficient to indicate four attacks at most, with a window size equal to 3, and three with a window size equal to 1. Experiments carried out for this paper include also the construction of *Self* with hyperspheres, and the detector generation for this purpose. Figure 7 presents detected anomalies for *sf* equal to 0.5 with $w = 1$ (Fig. 7a) and $w = 3$ (Fig. 7b). Because of the proposed way of computing $\boldsymbol{v}$ and different sliding window method, the results differ from those presented in [4], while in both cases with hyperspherical *Self* construction the system was unable to discover all five attacks. However, probe attacks were clearly indicated, what suggests, that the combination of these parameters works well with this kind of attack, regardless of *Self* design applied.

6.4 Experiment #4 – Coevolution Effectiveness for Randomly Generated Set of Anomalies

Mechanisms of coevolution have been tested to check, if there is a possibility of applying it to enhance the detector generation process. A set of 1,000 randomly generated vectors (Set A) from *Nonself* has been assumed as the second population

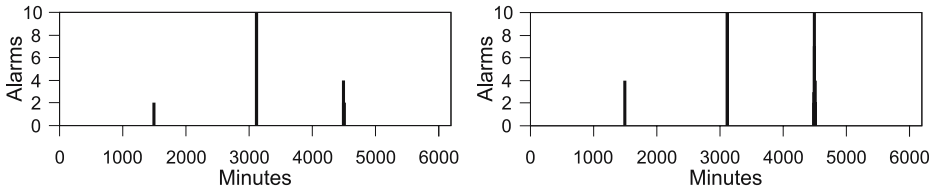

Fig. 7 Sphere detectors efficiency for $sf = 0.5$ and w equals **a** one and **b** three

coevolving with the population of generated detectors (*CoPopulation* – see Nonself Development Algorithm, Section 4). Set A was constructed as follows:

```
Self; SetA = null
DetectorSet = runNGA(Self)
for i=1 to 1000 do
   repeat
      anomaly = generateRandomState()
   while anomaly.isCapturedBy(Self)
   SetA.add(anomaly)
return SetA.
```

The detector generation process has been slightly altered for the coevolutionary NGA. After every run of NGA it was checked if all anomalies are already intercepted. If so, the algorithm was terminated. After the generation of a detector set using both coevolutionary and classic NGA, this set has been tested against the Set A, to check how many of elements have been covered.

Table 2 presents the results of the performed experiments, and the best of them have been highlighted. The detector sets have been developed for three different number of generations and for three different values of LTFE parameter. One additional detector set has been generated using standard NGA, without coevolution mechanism, marked with "—" symbol in LTFE column. The number of anomalies that have been covered differs, but the gain brought by coevolutionary NGA is insignificant, and for 500 generations classic NGA surpasses one with coevolution. These results can be explained by the random generation of the anomaly set. The distribution of states in *Nonself* is regular, and classic NGA, while trying to cover the largest space possible, intercepts also states generated without any specialization.

6.5 Experiment #5 – Coevolution Effectiveness for a Specialized Set of Anomalies

An alternative set of anomalies (Set B) has been generated and used as coevolving population in the coevolutionary mechanism. Set B was specialized as follows:

Table 2 Performance of detector sets for randomly generated set of anomalies

Generations	LTFE	Anomalies intercepted
100	5	866
	10	899
	20	**928**
	—	895
300	5	981
	10	962
	20	**982**
	—	972
500	5	963
	10	961
	20	973
	—	**987**

```
Self; SetB = null
DetectorSet = runNGA(Self)
for i=1 to 1000 do
   repeat
      anomaly = generateRandomState()
   while anomaly.isCapturedBy(Self)
    or anomaly.isCapturedBy(DetectorSet)
   SetB.add(anomaly)
return SetB.
```

The distribution of anomalies from Set B is irregular and some of its elements belong to areas, where detectors are harder to develop, for example, in small subspaces between *Self* structures. Results of the conducted experiments are presented in Fig. 8. The second experiment including coevolution shows a significant advantage of coevolutionary NGA and proves that the additional population can stimulate the detector generation process. Even a relatively small number (100) of generations allowed the coevolutionary NGA to obtain better results than the classic NGA with 750 generations and, consequently with less computational cost. Furthermore, one can notice that the efficiency of coevolution is LTFE dependent, but also depends on the number of generations, and results for more than 300 generations are worth further study. Another concern is a tradeoff between coevolution efficiency and detector fitness, calculated on the basis of various factors (see Section 4), which may cause worse performance of coevolutionary detectors development.

6.6 Experiment #6 – Coevolution Effectiveness for Less Restrictive Detector Overlay Criteria

The influence of detector overlay factor on the coevolution performance has been examined. The detector generation criteria were less strict, allowing overlying a certain percent of its volume between detectors. Set B was used for the coevolution

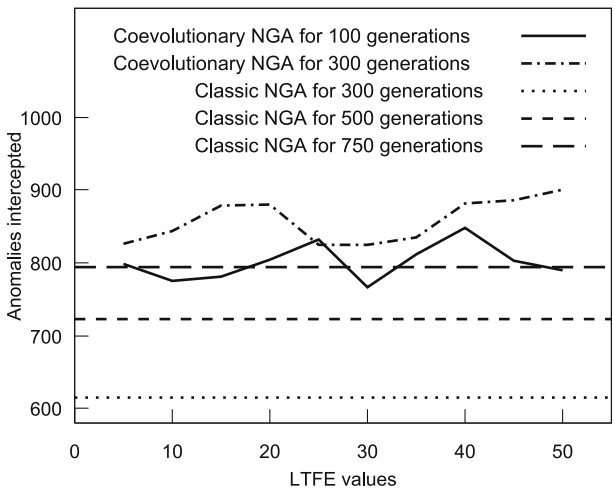

Fig. 8 Comparison of coevolution efficiency with classic NGA approach

mechanisms. Figure 9 illustrates dependencies between LTFE factor and the number of anomalies intercepted for three certain overlay tolerance levels. As one may see, efficiency of capturing anomalies rises with tolerance for other detectors in the set, what can be explained by the difference of goals between NGA and the coevolutionary mechanism. The first one tends to cover a space as large as possible, without overlaying already covered detectors. For the second one the goal is to capture some points enclosed in certain subspaces, and these are more important than the volume of detector. Therefore, the development of coevolutionary stimulated detectors that includes constraints of the classical NGA, restrains them from covering anomalies, if it would lead to overlap detector spaces.

It is worth noticing that the growing level of the overlay tolerance is related to the smoothness of lines, along with the efficiency of the detector set. It proves that a high sensitivity for overlapping of detectors may interfere and cause bad performance of coevolutionary stimulation in search for certain anomalies. Only three LTFE values are presented for tolerance level 0.75, showing the most significant peak for a given interval. These values seem to be optimal for the corresponding experiments presented below.

6.7 Experiment #7 – Initial Population Generation Process

Results presented in experiment #6 suggest, that some of the anomalies are impossible to reach for detectors overlapping with each other, even for a high tolerance level. This suggests, that unreachable points lay in vicinity of self structures, and cannot be captured because of the criterion of avoiding self space. To improve performance of NGA looking for nonself detectors, a new initial population generation mechanism has been developed. For every run of NGA, an initial population of detectors was generated in such a way, that none of them overlap with self space structures, or any detectors developed in previous runs. This process (henceforth called constrained generation) has been applied to the coevolutionary NGA to search for new detectors rather in uncovered space, and to lower the probability of the development of

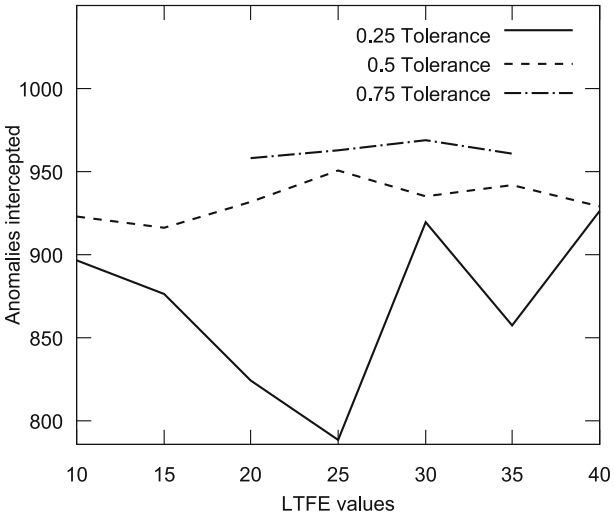

Fig. 9 Comparison of coevolution efficiency for detector overlay tolerance

improper detectors, overlapping with self, or detector structures. Figure 10 illustrates the performance of coevolutionary NGA based upon a constrained initial population, and the results for all three LTFE values are very similar. Although constrained generation offers greater efficiency, not entire Set B was covered. This indicates that some of anomalies are close enough to self states and still cause interferences for nonself detectors while intercepting them. Constraint generation seems to be a good method for narrowing the search space, but not sufficient. LTFE factor has small influence on interception process of the most difficult group, even taking into account a greatly increased fitness due to the captured anomalies. It plays no role if the detector covers *Self* space.

Vertebrate immune systems develop antibody detectors using libraries of genes, providing immature antigens with certain knowledge and information. This mechanism has been adapted to computer science domain as the gene libraries paradigm [3]. Our work introduces a simplified version of this method for improving the coevolution process, therefore to provide population of NGA with sufficient information. A different method of the initial population generation has been designed (henceforth called library generation), with only one library of information. It was constructed using unintercepted anomalies of Set B, treating them as information about the detector construction. As mentioned in Section 3, a detector is constructed using two vectors, *high* and *low*, and library generation process is based on a specific construction of the detectors, assuming for these vectors two randomly chosen, but different elements of library – anomaly vectors. Detectors constructed this way will not necessarily be correct, but infuse certain information into the population. This kind of a detector is constructed only with certain probability, which grows with a number of detectors developed, due to succeeding reduction of the anomalies set, and those most difficult that are left to find. Figure 11 presents the results of experiment involving library generation. Probability increment after single detector development was equal to 0.01.

As one may notice, the performance is improved by the library-generated population, and for the value of starting probability 0.3 and 0.35 it was possible to cover

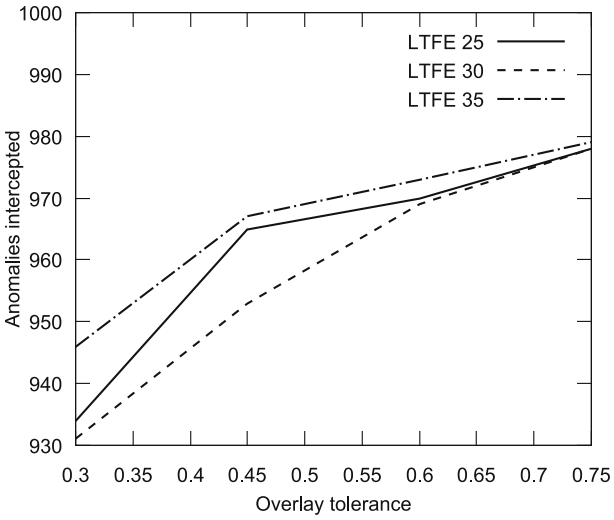

Fig. 10 Coevolution efficiency for constrained generation of detectors

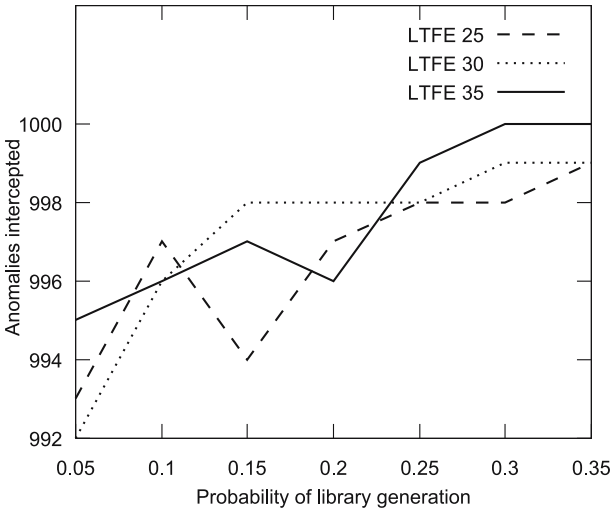

Fig. 11 Coevolution efficiency for library generation of detectors

all presented anomalies using developed detector sets. This mechanism seems to be efficient and using a properly constructed anomaly set it can boost the efficiency of entire detection system.

Figure 12 presents the results of comparing three different methods of initial population generation: unsupervised, constrained and library generation. All methods seem to have similar progress, but constrained generation method offers better results, while library generation is the best in terms of number of covered anomalies. Analysis of Fig. 12 indicates that a group of about 980 anomalies is easier to reach by generated detectors, and they present groups, that can be covered by a single detector. The group of anomalies possible to reach only by library-generated population is spread into subspaces that are hard to reach: every new detector manages to cover only few of them, and over half of all *Nonself* detectors developed with last method is covering these 20 anomalies. It proves that this particular coevolution

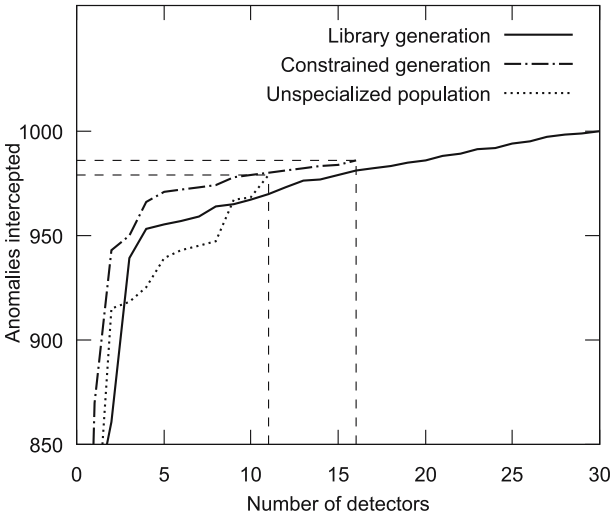

Fig. 12 Comparison of efficiency of three different initial population generation processes

method assures both effectiveness and accuracy, because a low number of detectors capture anomalies easier to cover, and those that are difficult to cover are captured as well, but with additional number of detectors.

7 Conclusion

The results of the conducted experiments indicate that the detectors generated by NGA proved to be effective, and hyperrectangular *Self* structures construction made a precise detection process possible. The presented approach is efficient and allows capturing all five kinds of simulated attacks in MIT data. While the hyperspherical design of *Self*, as presented by [4] made possible to indicate only four out of five attacks, and applied for *Self* development and NGA presented in this paper, three of them. It also has been shown that the coevolutionary mechanisms can enhance the detectors generation process and in the result can make detection process more effective against given patterns of attacks. Gathering data about some of those patterns in the form of coevolving sets can give in effect detector sets containing knowledge about attack subspaces. This mechanism can be compared to vaccine, which makes natural immune system more effective against certain illnesses.

The variability parameter $\boldsymbol{v}$ has been proven to be an important factor in the detector development process by influencing the *Self* volume. This parameter is responsible for adjusting the false alarms levels. Therefore, an algorithm for computing $\boldsymbol{v}$ from learning is very important in the attempt to improve the detection ability of a system, and the application of the statistical approach presented by [17] seems to be interesting in this context.

Further research may involve different parameter types and greater number of them. The analysis of the detection process data shows, that the system performs very effective in the case of Satan attack in a relatively short duration time, and has more problems with attacks like Portsweep or Neptune, although their duration last several times longer (see Table 1). Looking for dependencies between parameters and attack types is also a promising field of the research.

References

1. Axelsson, S.: Visualising intrusions: watching the webserver. In: Proceedings of the 19th International Information Security Conference (2004)
2. Beasley, D., Bull, D.R., Martin, R.R.: A sequential niche technique for multimodal function optimization. Evol. Comput. **2**(1), 101–125 (1993)
3. Cayzer, S., Smith, J., Marshall, J., Kovacs, T.: What have gene libraries done for AIS? In: Proceedings of the 4th International Conference on Artificial Immune Systems (2005)
4. Dasgupta, D., González, F.: An immunity-based technique to characterize intrusions in computer networks. IEEE Trans. Evol. Comput. **6**(3), 1081–1088 (2002)
5. Dozier, G.V., Brown, D., Hurley, J., Cain, K.: Vulnerability analysis of AIS-based intrusion detection systems via genetic and particle swarm red teams. In: Proceedings of the 2004 IEEE Congress on Evolutionary Computation (2004)
6. Eskin, M.: Anomaly detection over noiosy data using probability distributions. In: Proceedings of the 17th International Conference on Machine Learning. (2000)
7. Fawcett, T.: ROC graphs: Notes and practical considerations for data mining researchers. Technical Report HPL-2003-4 (2003)
8. Forrest, S., Perelson, A., Allen, L., Cherukuri, R.: Self–nonself discrimination in a computer. In: Proceedings of IEEE Symposium on Research in Security and Privacy (1994)
9. Garret, S.M.: How do we evaluate artificial immune systems? Evol. Comput. **13**(2) (2005)

10. Glickman, M., Balthrop, J., Forrest, S.: A machine learning evaluation of an artificial immune system. Evol. Comput. **13**(2), (2005)
11. Lee, W., Stolfo, S., Mok, K.: Mining in a data-flow environment: experience in network intrusion detection. In: Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining (1999)
12. Leon, E., Nasraoui, O., Gomez, J.: Anomaly detection based on unsupervised niche clustering with application to network intrusion detection. In: Proceedings of the 2004 IEEE Congress on Evolutionary Computation (2004)
13. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer, Berlin Heidelberg New York (1992)
14. MIT: http://www.ll.mit.edu/IST/ideval/index.html (1999)
15. Paredis, J.: Constraint satisfaction with coevolution. In: New Ideas in Optimization, McGraw-Hill, New York (1999)
16. Roesch, M.: Snort – lightweight intrusion detection for networks. In: Proceedings of the 13th Systems Administration Conference (1999)
17. Stibor, T., Timmis, J., Eckert, C.: A comparative study of real-valued negative selection to statistical anomaly detection techniques. In: Proceedings of the 4th International Conference on Artificial Immune Systems (2005)
18. Stibor, T., Timmis, J., Eckert, C.: On the appropriateness of negative selection defined over hamming shape-space as a network intrusion detection system. In: Proceedings of the 4th International Conference on Artificial Immune Systems (2005)
19. Wierzchon, S.T.: Artificial immune systems. Theory and application (in Polish). Warsaw, Poland: Exit (2001)