Efficient evolution of accurate classification rules using a combination of Gene Expression Programming and Clonal Selection

Vasileios K. Karakasis and Andreas Stafylopatis, *Member, IEEE*

Abstract— **A hybrid evolutionary technique is proposed for data mining tasks, which combines a principle inspired by the Immune System, namely the Clonal Selection Principle, with a more common, though very efficient, evolutionary technique, Gene Expression Programming (GEP).**

The clonal selection principle regulates the immune response, in order to successfully recognize and confront any foreign antigen, and at the same time allows the amelioration of the immune response across successive appearances of the same antigen. On the other hand, Gene Expression Programming is the descendant of Genetic Algorithms and Genetic Programming and eliminates their main disadvantages, such as the genotype-phenotype coincidence, though it preserves their advantageous features.

In order to perform the data mining task, the proposed algorithm introduces the notion of Data Class Antigens, which is used to represent a class of data. The produced rules are evolved by a clonal selection algorithm, which extends the recently proposed CLONALG algorithm. In the present algorithm, among other new features, a receptor editing step has been incorporated. Moreover, the rules themselves are represented as antibodies, which are coded as GEP chromosomes, in order to exploit the flexibility and the expressiveness of such encoding.

The proposed hybrid technique is tested on some benchmark problems of the UCI repository. In almost all problems considered, the results are very satisfactory and outperform conventional GEP both in terms of prediction accuracy and computational efficiency.

Key terms: Clonal Selection Principle, Gene Expression Programming, Artificial Immune Systems, Data Mining.

I. Introduction

Recently, the immune system and the mechanisms it utilizes in order to protect the body from invaders, has become a new promising field in the domain of machine learning. The natural immune system is a very powerful pattern recognition system, which has not only the ability to recognize and destroy foreign antigens, but also the ability to distinguish between its own and foreign cells. Additionally, the immune system can be characterized as a very effective reinforcement learning system, as it is capable of continuously improving its response to antigenic stimuli, which it has encountered in the past.

The mechanisms that regulate the behaviour of the natural immune system and how these mechanisms and concepts can be applied to practical problems is the matter of research in the field of Artificial Immune Systems (AIS). Early work on AIS examined its potential in machine learning and compared it to known techniques, such as artificial neural networks and conventional genetic algorithms (GAs) [9], [2], [3], [19], [20]. One of the first features of the natural immune system, which was modelled and used in pattern recognition tasks, was the Clonal Selection Principle, first introduced by Burnet [1] in 1959, upon which is based reinforcement learning in the immune system. Later research in immunology has enhanced Burnet's theory by introducing the notion of receptor editing [16], which will be discussed further in Section II-B. A first attempt to model the clonal selection principle was made by Weinard [22], though from a more biological point of view. Fukuda et al. [7] were the first to present a more abstract model of the clonal selection principle, which they applied to computational problems. However, it was the work of De Castro and Von Zuben [19], [4] on the CLONALG algorithm that considerably raised the interest around the clonal selection principle and its applications. CLONALG is an easy to implement and effective evolutionary algorithm, which may be applied both to optimization and pattern recognition tasks. CLONALG maintains a population of antibodies, which it evolves through selection, cloning and hypermutation. The most important features of CLONALG are that selection is a two-phase process and that cloning and hypermutation depend on the fitness of the cloned or mutated antibodies, respectively. Improvements and variations of CLONALG were later introduced by White and Garrett [23], who proposed the CLONCLAS algorithm, and by Nicosia et al. [15], who proposed a variation of the CLONALG, which used a probalistic half-life of B-cells and a termination criterion based on information theory. A more sophisticated application of the clonal selection principle is the AIRS [21] supervised learning system, which combines aspects of immune network theory [19] with the concept of the clonal selection principle.

In this work, we examine further an enhanced implementation of CLONALG that we have proposed in [10]. The innovative features of our approach may be summarized as follows: the memory update process is reviewed and defined in a more formal manner, providing some additional features. Antigens are no longer defined as symbol strings and the concept of *generic antigens* is introduced. Antibodies are defined as symbol strings of a language $\mathcal{L}$ and not as simple bit or real-valued vectors. Also, additional control is included in the proliferation phase of the algorithm and population refresh is altered. A new feature in our implementation is a step of receptor editing, which was added just before the first selection of antibodies. Receptor editing is expected to provide wider exploration of the solution space and helps the

The authors are with the School of Electrical and Computer Engineering, National Technical University of Athens, Zographou, Athens 157 80, Greece (phone: +30 210 772 2508, fax: +30 210 772 2109, email: andreas@cs.ntua.gr, bkk@cslab.ntua.gr).

algorithm avoid local optima [19].

The above enhanced clonal selection algorithm is coupled with a relatively new evolutionary technique, Gene Expression Programming (GEP), and used to mine classification rules in data. GEP [5] was introduced by Ferreira as the descendant of Genetic Algorithms and Genetic Programming (GP), in order to combine their advantageous features and eliminate their main handicaps. The most innovative feature of GEP is that it separates genotype from phenotype of chromosomes, which was one of the greatest limitations of both GAs and GP. In this paper, we isolate from GEP the representation of chromosomes, henceforth antibodies, and use the modified version of CLONALG to evolve them, so as to exploit its higher convergence rate. The actual classification of data and the formation of rules is based mainly on the work of Zhou et al. [24], who have successfully applied GEP to data classification. Specifically, the one-against-all learning technique is used in order to evolve rules for multiple classes and the Minimum Description Length (MDL) principle is used to avoid data overfitting. However, in contrast to Zhou et al., who use a two-phase rule pruning, we use only a prepruning phase through the MDL criterion, which in some cases yields more complex rulesets. Finally, the concept of Data Class Antigens (DCA) is introduced, which represents a class of data to be mined. Apart from generic antigens, a new multiple-point multiple-parent recombination genetic operator is added to GEP, in order to implement receptor editing. The proposed algorithm was tested against a set of benchmark problems and the results were very satisfactory both in terms of ruleset accuracy as well as in terms of the computational resources required.

The rest of the paper is structured as follows: Section II provides a quick overview of the clonal selection principle and its basic concepts. Section II-C describes our version of CLONALG. Section III provides a brief description of GEP and also introduces the multiple-point multiple-parent recombination operator. In Section IV it is described how the proposed hybrid technique is applied to data mining and Section V presents some experimental results on a set of benchmark problems. Finally, Section VI concludes the paper and proposes future work.

II. Overview of the Clonal Selection Principle

The clonal selection principle refers to the algorithm utilized by the immune system to react to an antigen. The clonal selection theory was originally proposed by Burnet [1] and establishes the idea that only those lymphocytes that better recognize the antigen are selected to be reproduced.

When an antigen invades the organism, the first immune cells to be activated are the T-lymphocytes, which have the ability to recognize the foreign organism. Once they have successfully recognized the antigen, they start secreting cytokines, which in turn activate the B-lymphocytes. After activation, B-lymphocytes start proliferating and finally mature and differentiate into plasma and memory cells. Plasma cells are responsible for the secretion of antigen-specific antibodies, while memory cells remain inactive during the current immune response; they will be immediately activated when the same antigen appears again in the future.

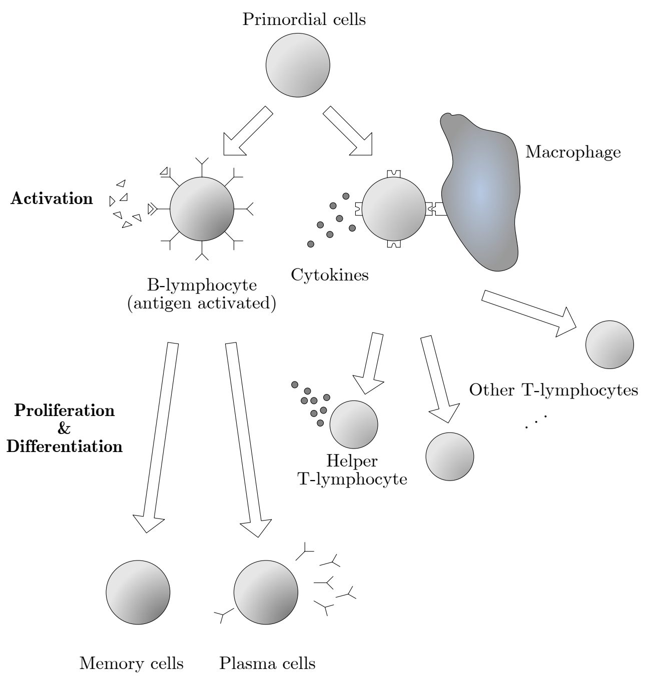


Fig. 1. The clonal selection principle.

The clonal selection principle can be summarized in the following three key concepts:

1) The new cells are clones of their parents and they are subjected to somatic mutations of high rate (hypermutation).
2) The new cells that recognize self cells are eliminated.
3) The mature cells are proliferated and differentiated according to their stimulation by antigens.

When an antigen is presented to the organism, apart from T-lymphocytes, some B-lymphocytes bind also to the antigen. The stimulation of each B-lymphocyte depends directly on the quality of its binding to the specified antigen, i.e. its affinity to the antigen. Thus, the lymphocytes that better recognize the antigen leave more offspring, while those that have developed self-reactive receptors or receptors of inferior quality are eliminated. In that sense, the clonal selection principle introduces a selection scheme similar to the natural selection scheme, where the best individuals are selected to reproduce. The clonal selection principle is depicted in Figure 1. In the following, learning in the immune system and the mechanisms for the immune response maturation are briefly described. A more detailed presentation can be found in [19], [4].

A. Learning in the immune system

During its lifetime an organism is expected to encounter the same antigen many times. During the first encounter, there exist no specific B-lymphocytes, and thus only a small number of them are stimulated and proliferate (*primary immune response*). After the infection is successfully treated,

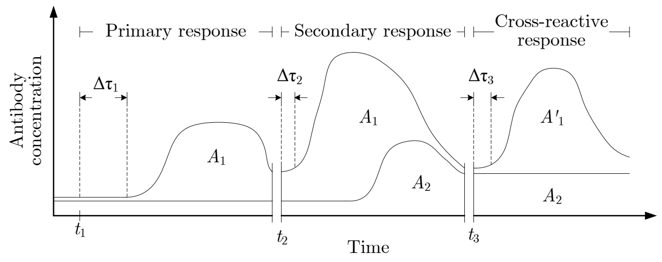


Fig. 2. Antibody concentration during the primary, the secondary and the cross-reactive immune response.

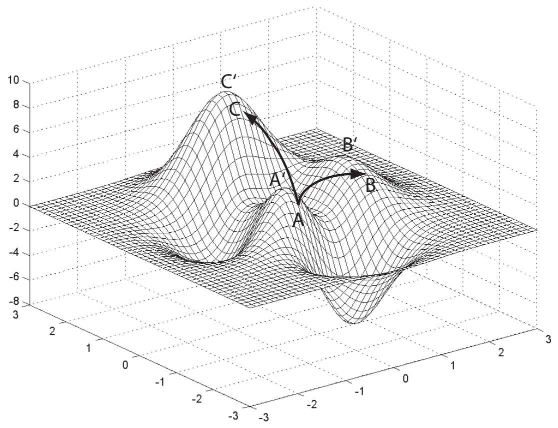


Fig. 3. Two dimensional representation of the antibody-antigen binding space. Hypermutation discovers local optima, whereas receptor editing can discover the global optimum.

the B-lymphocytes that exhibited higher affinities are kept in a memory pool for future activation. When the same antigen is encountered again in the future, memory B-lymphocytes are immediately stimulated and start proliferating (*secondary immune response*). During their proliferation, B-lymphocytes are subjected to a hypermutation mechanism, which may produce cells with higher affinities. After the suppression of the immune response, the best lymphocytes enter the memory pool. The process of "storing" the best cells into memory, may lead to the reinforcement of the immune responce across successive encounters of the same antigen, as better cells will always be the subject of the evolution. In that sense, the immune response could be considered as a reinforcement learning mechanism.

This notion could be schematically represented as in Figure 2, where the x-axis represents time and the y-axis represents the antibody concentration. In this figure, A_1, A_2 and A'_1 represent different antigens that are successively presented to the organism. When A_1 is first encountered at moment t_1, there exist no specific lymphocytes for this antigen, thus a lag phase ($\Delta\tau_1$) is introduced until the appropriate antibody is constructed. In moment t_2, A_1 appears again, along with the yet unknown antigen A_2. The response to A_2 is completely similar to primary response to A_1, which proves the specificity of the immune response, while the current response to A_1 is considerably faster ($\Delta\tau_2 \ll \Delta\tau_1$) and more effective (higher antibody concentration). The third phase depicted in Figure 2 reveals another important feature of the immune memory: it is an *associative* memory. At moment t_3 antigen A'_1, which is structurally similar to antigen A_1, is presented to the immune system. Although A'_1 has never been encountered before, the immune system responds very effectively and directly ($\Delta\tau_3 \approx \Delta\tau_2$). This can be explained by the fact that the A_1-specific antibodies can also bind to the structurally similar A'_1, hence providing a qualitative basis for the A'_1 antibodies, which leads to a more effective immune response. This type of immune response is called *cross-reactive immune response*.

The gradual amelioration of the immune response, which is achieved through successive encounters of the same antigen, is described by the term *maturation of the immune response* or simply *affinity maturation*. The mechanisms through which this maturation is achieved, are described in Section II-B.

B. Affinity maturation mechanisms

The maturation of the immune response is basically achieved through two distinct mechanisms:

1) hypermutation, and
2) receptor editing.

Hypermutation introduces random changes (mutations) with high rate into the B-lymphocyte genes, that are responsible for the formation of the antibodies' variable region. The hypermutation mechanism, apart from the differentiation of the antibody population, permits also the fast accumulation of beneficial changes to lymphocytes, which in turn contributes to the fast adaptation of the immune response. On the other hand, hypermutation, due to its random nature, may often introduce deleterious changes to valuable lymphocytes, degrading thus the total quality of the antibody population. Therefore, there must exist some strict and efficient mechanism for the regulation of the hypermutation mechanism. Such a mechanism would allow mutations with high rate to lymphocytes that produce poor antibodies, and impose a very small or null rate to lymphocytes with "good" receptors.

Receptor editing was introduced by Nussenzweig [16], who stated that B-lymphocytes undergo also a molecular selection. It was discovered, that B-lymphocytes with low quality or self-reactive receptors destroy those receptors and develop completely new ones by a V(D)J recombination. During this process, genes from three different gene libraries (libraries V, D and J) are recombined in order to form a single gene in the B-lymphocyte genome, which is then translated into the variable region of antibodies. Although this mechanism was not embraced in Burnet's clonal selection theory, it can be easily integrated as an additional step before the final selection of lymphocytes.

The existence of two mechanisms for the differentiation of the antibody population is not a redundancy, but in contrast they operate complementarily [19]. As depicted in Figure 3, the hypermutation mechanism can only lead to local optima of the antibody-antigen binding space (local optimum A'), whereas receptor editing can provide a more global exploration of the binding space ("jumps" to B and C). Thus, hypermutation can be viewed as a refinement mechanism, which—in combination with receptor editing, that provides a coarser but broader exploration of the binding space—can lead to the global optimum.

C. An implementation of the clonal selection principle

The basis of the hybrid evolutionary technique presented in this paper is an implementation of CLONALG, which was first introduced by Von Zuben and De Castro [4], [19]. Although the basic concept of the algorithm remains the same, the implementation presented here, which is henceforth called ECA (Enhanced Clonal Algorithm), is built upon a slightly different theoretical background, in order to be easily coupled with the GEP nomenclature. Additionally the following features were added or enhanced in ECA:

- A receptor editing step was added just before the first selection of antibodies, in order to achieve better exploration of the antibody-antigen binding space.
- The update process of the population memory is defined in a more formal manner.
- Antigens are no longer defined as simple symbol strings. The concept of *generic antigens* is instead introduced, which allows application of the algorithm to a variety of machine learning problems.
- Antibodies are represented as symbol strings and not as bit strings or real-valued vectors.
- Cloning of best cells depends also on the number n_b of clones selected in the first selection phase. This allows a finer and more accurate control over the clones produced, as two variables (n_b and clone factor) control the cloning process.
- The algorithm is more configurable than the pure CLONALG. Specifically, it allows more memory cells to recognize a single antigen and more memory cells to be updated simultaneously. During the population refresh phase, some improved clones are allowed to enter the population, replacing some poor existing members. This last feature is also common to CLONCLAS, which is an enhanced implementation of CLONALG presented in [23].

ECA maintains a population $\mathcal{P}$ of antibodies[1], which are the subject of evolution. An antibody is defined to be a string of a language $\mathcal{L}$, such that

$$\mathcal{L} = \{s \in \Sigma^* \text{ and } |s| = l,\ l \in \mathbb{N}\},$$

where Σ is a set of symbols and l is the length of the antibody. Both Σ and l are parameters of the algorithm and are set in advance.

[1]In the remaining of the paper no distinction will be made between antibodies and lymphocytes, as the former constitute the gene expression of the latter.

The population of antibodies, $\mathcal{P}$, can be divided into two distinct sets, $\mathcal{M}$ and $\mathcal{R}$, such that

$$\mathcal{M} \cup \mathcal{R} = \mathcal{P} \text{ and } \mathcal{M} \cap \mathcal{R} = \emptyset,$$

where $\mathcal{M}$ contains the memory cells and $\mathcal{R}$ contains the remaining cells.

A set $\mathcal{G}$ of antigens to be recognized is also defined. It is worth mentioning that the only restriction imposed on $\mathcal{G}$, is that it should be a collection of uniform elements, i.e. elements of the same structure or representation; no assumption is made as to the structure or the representation themselves. For that reason, these antigens are called *generic antigens*. Generic antigens may allow CLONALG to be used in a variety of pattern recognition problems.

Between sets $\mathcal{G}$ and $\mathcal{M}$ a mapping K is defined, such that

$$K : \mathcal{G} \to \mathcal{M}.$$

This mapping associates antigens with memory cells, which are capable of recognizing them. Generally, the mapping K is not a function, as a single antigen may be recognized by a set of different memory cells, or stated differently, a set of memory cells may be "devoted" to the recognition of a specific antigen. For example, let $\mathcal{G} = \{g_0, g_1\}$ and $\mathcal{M} = \{m_0, m_1, m_2\}$, then a mapping K, which is defined as

$$K(g_0) = m_0,$$
$$K(g_1) = m_1,$$
$$K(g_1) = m_2,$$

states that memory cell m_0 recognizes antigen g_0, and memory cells m_1, m_2 recognize antigen g_1. The fact that K is not a function, may impose a difficulty during the phase of memory updating, since it will not be clear to the algorithm which memory cell to replace. For that reason, apart from the mapping K, a memory update policy will be needed in order to update memory in a consistent manner. This policy is responsible for selecting the memory cells, which will be candidate for replacement, and the way this replacement will take place, as it is possible that more than one memory cells are updated during the memory update phase (see algorithm step 8 below). An important point in the implementation of ECA, is that both the size of the memory set $\mathcal{M}$ and the mapping K are determined in advance and remain unchanged throughout the execution of the algorithm. That means that a specific antigen will always be recognized by a specific set of memory cells, or, from a computational point of view, a specific antigen will always be recognized by cells in specific memory positions.

The mapping K divides the memory set $\mathcal{M}$ into a set of distinct subsets $\mathcal{M}_i$ such that

$$\mathcal{M}_i = \{m : m = K(g), g \in \mathcal{G}\}, \quad 1 \leq i \leq n = |\mathcal{G}|.$$

If $\bigcup_{i=1}^{n} \mathcal{M}_i = \mathcal{M}$, then K defines a partition over the set $\mathcal{M}$ and $\mathcal{M}$ is called minimal, as every memory cell recognizes an antigen. It can easily be proved that a population set $\mathcal{P}$

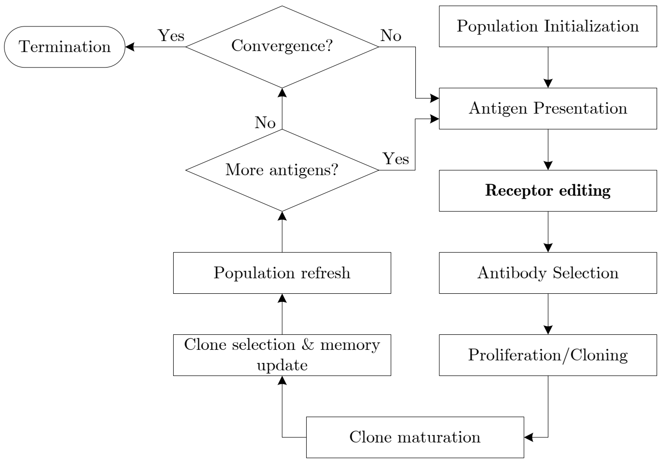


Fig. 4. The ECA algorithm; a modified version of the CLONALG algorithm.

with a non minimal memory set can always be converted to an equivalent population set $\mathcal{P}'$ with a minimal memory set.

Finally, the affinity function between antibodies and antigens is defined as

$$f : \mathcal{L} \times \mathcal{G} \rightarrow \mathbb{R}.$$

The function f should return higher values when an antibody binds well to an antigen and lower ones, when the binding is inadequate. Usually, f is normalized in the interval $[0, 1]$. The way the "binding" between an antibody and an antigen is defined, depends mainly on the representation of the antigens and the semantics of the antibody language $\mathcal{L}$, which makes this concept rather problem specific. The definition of "binding" for the problem of data mining considered in this paper is presented in Section IV-B.

Having described the theoretical background of our version of ECA, a more detailed description of the algorithm follows (see also Figure 4).

Step 1. [Population initialization] Each member of the population is initialized as a random string of language $\mathcal{L}$. Additionally, a temporary set $\mathcal{G}_r$ is defined such that $\mathcal{G}_r = \mathcal{G}$.

Step 2. [Antigen presentation] An antigen g_i is selected randomly from the set $\mathcal{G}_r$ and is presented to the population. For each member of the population, the affinity function f is computed and the affinity measure produced is assigned to that member. Finally, antigen g_i is extracted from set $\mathcal{G}_r$.

Step 3. [Receptor editing] The n_e antibodies with the lower affinities are selected to undergo the receptor editing process. The best n_p antibodies are also selected to form a gene pool, from which genes will be drawn during the V(D)J recombination. The exact procedure of the V(D)J recombination is described later in this section.

Step 4. [Selection of best antibodies] The best n_b antibodies, in terms of their affinity, are selected and form the set $\mathcal{B}$.

Step 5. [Proliferation of best antibodies] Each antibody of the set $\mathcal{B}$ is cloned according to its affinity. Generally, antibodies with higher affinities produce more clones. The set of clones is called $\mathcal{C}$.

Step 6. [Maturation of the clones] Each clone c_j of set $\mathcal{C}$ is mutated at a rate a_j, which depends on the affinity of the clone. Generally, antibodies with higher affinities should be mutated at a lower rate. The mutated clones form the set $\mathcal{C}_m$.

Step 7. [Affinity of clones] The antigen g_i is presented to the set of mutated clones, $\mathcal{C}_m$, and the affinity function f is computed for each clone.

Step 8. [Memory update] The n_m best mutated clones are selected according to their affinity, and form the set $\mathcal{B}'$. The mapping K is then applied to antigen g_i, and the set $\mathcal{M}_i$ of memory cells that recognize g_i is obtained. Next, the memory update policy is applied and a set $\mathcal{M}_i'$, such that $|\mathcal{M}_i'| = n_m \leq |\mathcal{M}_i|$ is produced, which is the set of the candidate for replacement memory cells. These cells will be replaced by selected clones with higher affinities, so that at the end of this process the following inequality holds:

$$f(m, g_i) \geq f(a, g_i), \quad \forall m \in \mathcal{M}_i', \forall a \in \mathcal{B}'.$$

The way the replacement will take place, i.e. how the selected memory cells will be replaced by the best clones, is also a matter of the memory update policy described below.

Step 9. [Population refresh] At this step, the population is refreshed in order to preserve its diversity. Refreshing may be performed in two distinct ways. Either n_r cells are randomly selected from the set of mutated clones and are inserted into the population replacing some existing cells, or the n_d worst cells, in terms of their affinity to antigen g_i, are replaced by completely new ones, which are random strings of language $\mathcal{L}$.

Step 10. [End conditions check] If $\mathcal{G}_r \neq \emptyset$, then the algorithm is repeated from Step 2. Otherwise, the satisfaction of a convergence criterion between the memory and the antigen set is checked. At this point, an evolution generation is said to be complete. If no convergence has been achieved, then $\mathcal{G}_r \leftarrow \mathcal{G}$ and the algorithm is repeated from Step 2, otherwise the algorithm is terminated. ∎

1) Proliferation control and regulation of the hypermutation mechanism: The success of the ECA algorithm in a pattern recognition problem depends heavily on the regulation of the proliferation of the best antibodies and the maturation of the clones. Thus, a control mechanism should be established, that could firstly augment the possibility that a "good" clone will appear, and secondly guarantee to the most possible extent that the already "good" clones will not disappear.

The ECA algorithm uses almost the same control mechanisms as CLONALG. Namely, in order to control the proliferation of the best antibodies, it first sorts the set $\mathcal{B}$

of best antibodies in descending order, and then applies the formula

$$n_i = \text{round}\left(\frac{\beta \cdot n_b}{i}\right), \quad 1 \le i \le n_b \tag{1}$$

to compute the number of clones that each antibody will produce. In this formula, $\text{round}(\cdot)$ is the rounding function, β is a constant, called *clone factor*, n_b is the total number of antibodies selected in Step 4 of the algorithm, and i is the rank of each selected antibody in the ordered set $\mathcal{B}$. What is important here is that the number of clones depends on the number of antibodies selected before cloning and not on the total size of population as in CLONALG. This allows finer control over the proliferation of the best clones, which may lead to better resource utilization.

Hypermutation is controlled through the exponential function

$$\alpha(x) = \alpha_{\max} e^{-\rho \cdot x}, \quad \alpha_{\max} \le 1, \tag{2}$$

where α is the mutation rate, $\alpha_{\max}$ is a maximum mutation rate, ρ is a decay factor, and x is the affinity normalized in the interval $[0, 1]$.

2) Memory update policy: In general, the memory update policy depends directly on the cardinality of the memory and antigen sets, the mapping K and the number of best clones, n_m, candidate for entering the memory pool. In the problem at hand, a rather simple mapping K and a straightforward memory update policy were used. First, we require that $|\mathcal{M}| = |\mathcal{G}|$ and the mapping K is defined to be a one-to-one mapping:

$$\mathcal{M} = K(\mathcal{G}).$$

In the implementation presented here only one clone is allowed to enter the memory in each generation and therefore, the memory update policy is straightforward: the cell to be replaced is the one denoted by the mapping K, or—stated differently—it holds that

$$\mathcal{M}'_i = \mathcal{M}_i, \quad 1 \le i \le |\mathcal{G}|.$$

Finally, as a convergence criterion between memory and the antigen set, the Least Mean Square (LMS) criterion is used, considering

$$e = \sum_{i=1}^{|\mathcal{G}|} (m_i - g_i)^2, \quad m_i = K(g_i).$$

3) Receptor editing: The notion behind the implementation of the receptor editing process is to form new antibodies from random substrings of different antibodies of reasonably high quality. For that reason, during the receptor editing process, the n_p best antibodies are selected to form a pool of genes. A gene is considered to be any substring of an antibody[2]. These genes will be recombined through the V(D)J recombination, in order to form the new antibody. V(D)J recombination is a five step process, which is described by the following algorithm (see also Figure 5), where l_c is the current length of the antibody under construction, l_g is the length of the gene selected, and L is the length of the entire antibody.

[2]The reference to “gene” should not be confused with a GEP gene.

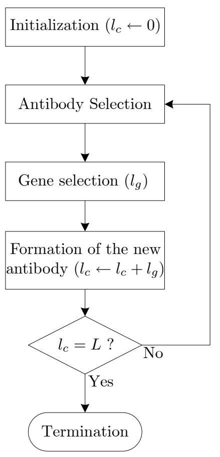

Initialization ($l_c \leftarrow 0$)
Antibody Selection
Gene selection (l_g)
Formation of the new antibody ($l_c \leftarrow l_c + l_g$)
$l_c = L$?
No
Yes
Termination

Fig. 5. An implementation of the V(D)J recombination.

Step 1. [Initialization] $l_c \leftarrow 0$.

Step 2. [Antibody selection] An antibody is selected randomly from the pool of antibodies.

Step 3. [Gene selection] A substring of random length l_g is selected from the selected antibody. The length l_g should conform to the restriction

$$l_g \le L - l_c.$$

Step 4. [Antibody formation] The gene selected is appended to the new antibody, and the length l_c is updated:

$$l_c \leftarrow l_c + l_g$$

Step 5. [End condition] If $l_c = L$, then the algorihtm terminates. Otherwise, it is repeated from Step 2. ∎

D. Convergence analysis of ECA

The ECA algorithm has plenty of parameters and, as a result, its tuning may be rather tedious. In this section, a primary approach toward managing the algorithm parameters will be performed. A character recognition problem will be used in order to examine some of the main parameters of ECA, and how these affect its convergence. The character recognition problem consists of 8 characters, as depicted in Figure 6 [11]. In this step of analysis, receptor editing is disabled and we seek to understand how the remaining ECA parameters affect convergence. More specifically, we will examine how mutation rate decay and clonal expansion affect performance and accuracy. We will assume that the algorithm converges if and only if it does so within 200 generations. As a convergence criterion, we use the Mean Squared Error (MSE) criterion, formally defined as

$$e = \frac{1}{n} \sum_{i=1}^{n} d_i^2, \tag{3}$$

where d_i is the normalized Hamming distance between the memory cells and the presented antigens or patterns. In

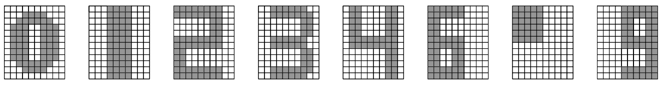


Fig. 6. The Lippman character set.

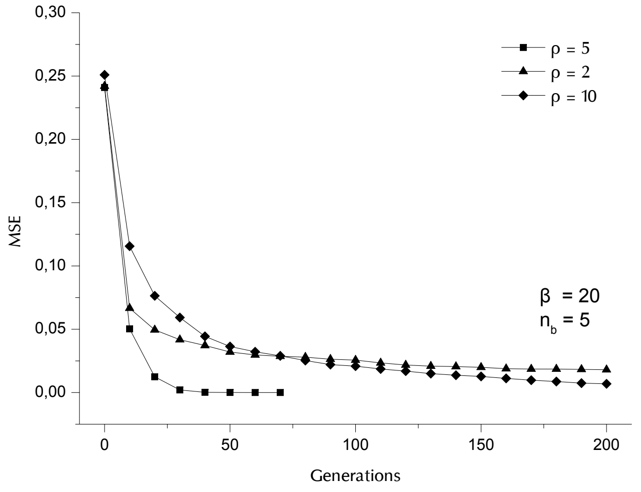


Fig. 7. ECA convergence rate relative to mutation rate decay ρ.

the character recognition benchmark problem we impose a minimum MSE of 10^{-3}.

One critical parameter of ECA, which can considerably affect convergence, is mutation rate decay ρ, as depicted in Figure 7, where the MSE is plotted against generation number. When $\rho = 5$, ECA converges rather fast within about 70-75 generations, but, when $\rho = 2$ or $\rho = 10$, the algorithm does not succeed to converge within the window of 200 generations. When $\rho = 10$, the new antibodies are quite similar to those of previous generations, so it will take the algorithm longer to form a set of memory cells of adequate quality. The exact opposite happens when $\rho = 2$. The high mutation rate imposed to new antibodies in early generations will soon create a set of quality cells. However, while these high mutation rates are beneficial at the beginning, they tend to hinder the overall performance of the algorithm through generations, as they may insert deleterious changes to quality cells obtained so far. The algorithm will eventually converge, because the best cells are always kept in memory, but at a very slow rate. This hindering behaviour of high mutation rates can be also deducted from Figure 7, where the $\rho = 10$ curve gets lower than the $\rho = 2$ curve from generation 70 and forth.

Another critical parameter of the algorithm is the product βn_b, which controls the creation of clones (see Equation 1). As depicted in Figure 8, larger values of βn_b lead to faster convergence, although differences are rather small as this product increases. This behaviour could be explained by the fact that the more the clones, the better the chances for a quality antibody to appear. However, after a certain number of clones is attained, more out of them would not benefit at all, because they are already numerous enough to accomodate any beneficial mutation introduced for a given mutation rate.

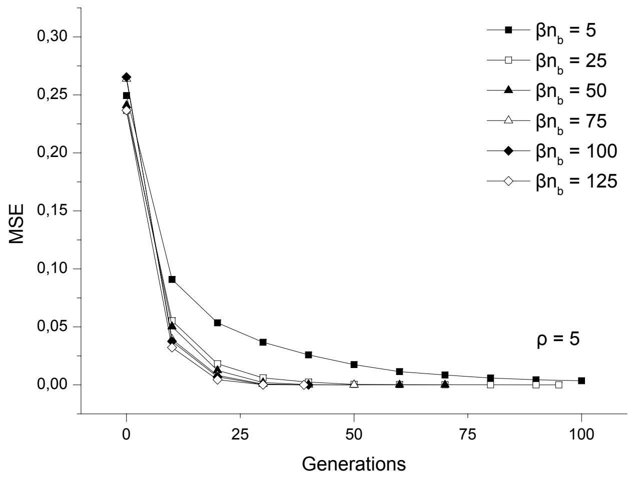


Fig. 8. Convergence rate relative to the product βn_b.

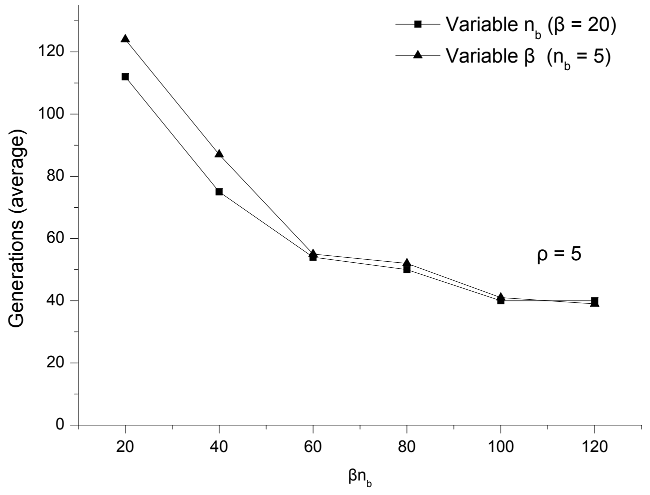


Fig. 9. How independent variation of β and n_b affect convergence.

This fact imposes a subtle tradeoff between convergence speed and computational resources needed, as more clones would not improve convergence rate, but, in contrast, they would reduce the overall performance of the algorithm.

Finally, another interesting issue concerns whether and how convergence rate is affected by separately modifying β or n_b while keeping the βn_b product constant. In Figures 9 and 10, the average convergence rate and its standard deviation are plotted against the product βn_b. Each figure displays two curves, one corresponding to varying n_b, while keeping β constant ($\beta = 20$), and the other corresponding to varying β, while keeping n_b constant ($n_b = 4$). Although average convergence rate seems not to be influenced by β and n_b separately, especially for higher values of their product, the choice of β and n_b seems to affect the standard deviation of convergence rate, and hence stability of the algorithm.

III. GEP ANTIBODIES

In the hybrid data mining approach presented here, antibodies are represented as GEP chromosomes, and henceforth will be referred to as GEP antibodies, in order to be

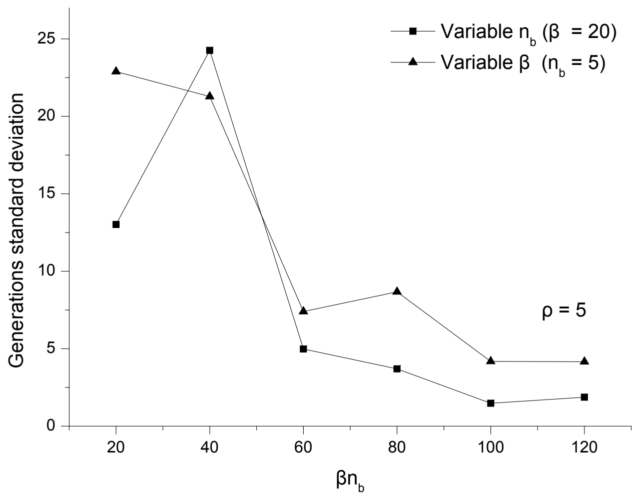


Fig. 10. How independent variation of β and n_b affect standard deviation of convergence.

```
0123456789012345678901234567890
/aQ/b*ab/Qa*b*-ababaababbabbbba
```

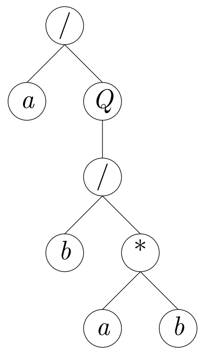


Fig. 11. Translation of a GEP gene into an ET.

distinguished from classical linearly encoded and expressed antibodies. GEP antibodies may not be considered as fully functional GEP chromosomes, in that they do not support all the genetic operators (see Section III-B) defined by GEP. Nonetheless, such support could be easily integrated, as GEP antibodies maintain the exact structure of GEP chromosomes.

Gene Expression Programming was first introduced by Ferreira [5], [6] as the descendant of Genetic Algorithms (GAs) and Genetic Programming (GP). It fixes their main disadvantage, the genotype-phenotype coincidence, though preserving their main advantages, namely the simplicity of the GAs' chromosome representation and the higher expression capabilities of GP. This dual behaviour is achieved through a chromosome representation, which is based upon the concepts of *Open Reading Frames (ORFs)* and non-coding gene regions, which are further discussed in the following section.

A. The GEP antibody genome

The GEP genome is a symbol string of constant length, that may contain one or more genes linked through a linking function. A GEP gene is the basic unit of a GEP genome, and consists of two parts: the *head* and the *tail*. In the head, any symbol, either terminal or function symbol, is allowed. In the tail, only terminal symbols are allowed. The length of the tail depends on the actual length of the head of the gene, according to the formula [5]

$$t = h(n-1)+1, \tag{4}$$

where t is the length of tail, h is the length of head, and n is the maximum arity of the function symbols in the GEP alphabet. This formula guarantees that the total gene length will be enough to hold any combination of function symbols in the head of the gene, while at the same time preserving the validity of the produced expression.

To better illustrate the concepts of GEP, consider the following example. Let $F = \{Q, *, /, -, +\}$ be the function symbol set, where Q is the square root function, and $T = \{a, b\}$ be the terminal symbol set. Let also $h = 15$, and thus, from equation (4), $t = 16$, as the maximum arity of function symbols is $n = 2$, which is the arity of $*, /, -$ and $+$. Consider, finally, the following GEP gene with the above characteristics (the gene tail is denoted in an italic font):

```
0123456789012345678901234567890
/aQ/b*ab/Qa*b*-ababaababbabbbba
```

This gene is decoded to an expression tree (ET), as depicted in Figure 11. The decoding process is rather straightforward: the ET is constructed in a breadth-first order, while the gene is traversed sequentially. The expansion of the ET stops when all leaf nodes are terminal symbols. However, the most important issue in the decoding process is that only a part of the GEP gene is translated into an expression tree. This part is called an *Open Reading Frame (ORF)* and has variable length. An ORF starts always at position 0 and spans through to the position where the construction of the corresponding ET has finished. The rest of the GEP forms the *non-coding region*.

Using such a representation of genes, it is obvious that GEP distinguishes the expression of genes, i.e. their phenotype, from their representation, i.e. their genotype. Additionally, it succeeds in a rather simple and straightforward manner to couple the higher expression capabilities of expression trees and the effectiveness of the pure linear representation.

Finally, GEP antibodies may contain multiple genes. In such a case, each gene is translated independently and they are finally combined by means of a linking function [6]. The structure of a multigene antibody is depicted in Figure 12.

B. Genetic operators

The flexibility of GEP antibodies allows the easy adoption of almost any genetic operator, that is used by GAs. The only additional requirement is that these operators should preserve the structure of the GEP genes, i.e. no operator may insert non terminal symbols in the gene tails.

For the purposes of this work, only two genetic operators are used: the mutation operator, as was originally defined for

```
012345678901234
+bQ**b+bababbbb
--b/ba/aaababab
*Q*a*-/abaaaaab
```

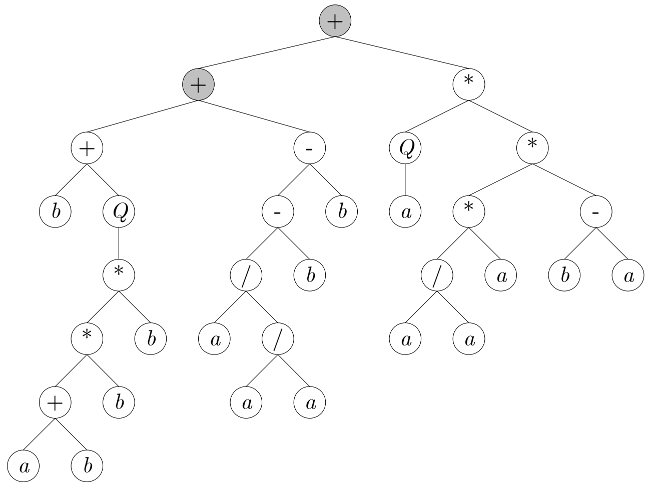


Fig. 12. A multigene antibody with 3 genes. Individual genes are linked through addition.

GEP [5], [6], and a multiple-parent multiple-point recombination operator, which is introduced here.

The mutation operator, which is used to perform the hypermutation of antibodies, is rather trivial and is not presented here. The multiple-parent multiple-point recombination operator is analogous to one- and two-point recombination operators used by the standard GEP, with the difference that more than two parents are used and more than two gene split positions are allowed. This operator was introduced in order to support the V(D)J recombination mechanism, which was presented in Section II-C, and to provide a better exploration of the solution space, as well.

The way this operator acts over GEP antibodies resembles the way V(D)J recombination is performed. Initially, n antibodies are randomly selected to form the set of parents. The number of parents, n, is a paremeter of the algorithm. Next, a split point in every parent is randomly generated. Split points should be in ascending position order, i.e. the split point of a parent antibody should be at the right of the split point of the previously splitted parent. Split point generation is repeated until a split point coincides with the end of a parent antibody. If all parents are split once and the last split point has not reached the end of an antibody, the split operation proceeds to the first splitted parent antibody adding a new split point to it. In that way, the parent antibodies are splitted multiple times. The offspring of this recombination is an antibody consisting of the gene segments between split points of their parents.

The multiple-parent multiple-point recombination could be better illustrated by the following example. Consider the antibody alphabet $\Sigma = \{Q, *, /-, +, a, b\}$, where Q is the square root function, $*, /, +, -$ are as usual, and a, b are terminal symbols. Let also $h = 5$ and $n = 3$. Finally, assume that the selection process yields the following three antibodies:

```
0123456789001234567890012345678900
Q+bb*bbbaba-**--abbbaaQ*a*Qbbbaab
/-++QbababbQ**abbabbaaQ*ab+abaaab
-+Qbabaaabb/Q*+aababbab*+*Qaaabab
```

If the set of the generated split points is $P = \{(6,1,1),(2,2,2),(9,2,3),(3,3,1),(10,3,2)\}$, where the triplet (i,j,k) signifies a split point at position i of the j-th gene of the k-th parent antibody, then the resulting gene segments are the ones underlined in the above scheme. The offspring of this recombination will be the combination of these 5 gene segments:

```
0123456789001234567890012345678900
Q+bb*bababbQ**+aababaaQ*ab+abaaab
```

The multiple-parent multiple-point recombination may offer considerable benefits to population diversity, as it mimics in a rather consistent manner the process of the natural V(D)J recombination.

IV. Application to data mining

In this section, the ECA algorithm and the basic representation concepts of GEP described above are combined, in order to be applied in data mining problems. Additional issues, such as antigen representation, affinity function and data covering algorithm, as well as overfitting avoidance and generation of the final rule set, are also treated in more detail in this section.

A. Antigen representation

The ECA algorithm is a supervised learning technique, where antigens play the role of patterns to be recognized. In a data mining task, a description of the data classes may represent the patterns for recognition. For that reason the concept of Data Class Antigens (DCAs) is introduced. A DCA represents a single data class of the problem and consists of a sequence of data records, which belong to the same class. DCAs conform to the generic antigen definition introduced in Section II-C, where antigens must be represented as a sequence of arbitrary objects of similar structure. In order to fully integrate the notion of DCAs into the ECA algorithm, an appropriate "binding" between DCAs and GEP antibodies should be defined, as well as a consistent affinity measure. These issues are treated in the next two sections.

B. Data Class Antigen recognition

A GEP antibody is said to better recognize a DCA, when it can produce a better classification of its instances. This is equivalent to saying that the best GEP antibody would be the one that identifies all instances of the class represented by the DCA as being instances of this actual class, provided no noise in data is present.

The classification technique used in this work is based on *one-against-all learning*, where an instance of a class is recongized against all other classes. This technique can be easily implemented using GEP antibodies, by classifying a record of a DCA in the class represented by this DCA, if the ET of the GEP antibody is evaluated positively. More strictly, this classification mechanism can be described by the following definition:

Definition IV.1. *A record* $\mathbf{r}$ *of a DCA* g*, which represents a data class* $\mathcal{C}_g$*, will be classified in this class by a GEP antibody, which is translated in the expression* P*, if and only if* $P(\mathbf{r}) > 0$*. Otherwise, it is not classified.*

This definition "binds" GEP antibodies to DCAs and makes the application of the ECA algorithm to a data classification problem rather straightforward; data classes of the problem are coded as DCAs, which are successively presented to ECA, which in turn evolves the GEP antibody population, in order to produce a rule set for the specified data class.

Every GEP antibody may be considered as a rule that describes the data of a DCA. A record or example $\mathbf{r}$ satisfies a rule R coded in GEP format, or more simply $\mathbf{r}$ is a positive example, if $P(\mathbf{r}) > 0$, where P is the expression represented by rule R. Otherwise, the example is considered negative. Similarly the coverage of a rule R is defined to be the set of all positive examples.

C. Affinity function and covering algorithm

Having defined the binding between GEP antibodies and DCAs in the previous Section, it is obvious that a rule of good quality will be one that covers as many positive examples and as few negative examples as possible. Instead of using pure rule completeness and consistency measures, a measure combining both rule completeness and consistency gain was used, as in [24]. More precisely, the affinity function is defined by the formula

$$f(R) = \begin{cases} 0, & \mathrm{consig}(R) < 0 \\ \mathrm{consig}(R) \cdot e^{\mathrm{compl}(R)-1}, & \mathrm{consig}(R) \geq 0 \end{cases}, \tag{5}$$

where $\mathrm{consig}(R)$ is the consistency gain of the rule R and $\mathrm{compl}(R)$ is the completeness of rule R, which can be defined as [12], [24]:

$$\mathrm{compl}(R) = \frac{p}{P}, \tag{6}$$

$$\mathrm{consig}(R) = \left(\frac{p}{p+n} - \frac{P}{P+N}\right)\frac{P+N}{N}, \tag{7}$$

In the above equations, p is the number of positive examples covered by rule R, n is the number of negative examples covered by rule R, P is the total number of positive examples, i.e. all examples belonging to the class under consideration, and N is the total number of negative examples, i.e. all the counter-examples of the class under consideration. It is easy to prove, that this affinity function favors rules with greater consistency rather than rules with high completeness. The use of a consistency gain measure, instead of a pure consistency one, was preferred, because the consistency gain actually compares the consistency of a prediction rule to a totally random prediction [12]. This is the reason why the affinity function f is set to 0 every time $\mathrm{consig}(R)$ is negative,

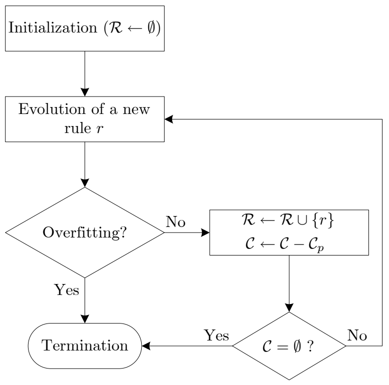

Initialization ($\mathcal{R} \leftarrow \emptyset$)
Evolution of a new rule r
Overfitting?
No
$\mathcal{R} \leftarrow \mathcal{R} \cup \{r\}$
$\mathcal{C} \leftarrow \mathcal{C} - \mathcal{C}_p$
Yes
Termination
Yes
$\mathcal{C} = \emptyset$?
No

Fig. 13. The covering algorithm used for the coverage of all positive examples of a data class. Overfitting detection is not presented in this figure.

which signifies that the rule R is worse than a random guess. Finally, f is normalized in the interval $[0, 1]$.

Covering algorithm: In real-life problems, a single rule is usually not adequate to describe a class of data. For that reason multiple rules are evolved for each class, so as to cover as many positive examples of the class as possible, avoiding, if possible, data overfitting. The covering algorithm used is rather simple and is briefly described in [24]. For each class in the problem one rule is firstly evolved, using the affinity criterion described above. If this rule fails to cover all positive examples of the class, then the covered examples are removed and another rule is evolved on the remaining examples. This process continues until all positive examples are covered or data overfitting occurs (see next). This algorithm is depicted in Figure 13, where $\mathcal{R}$ is the rule set under construction, $\mathcal{C}$ is the class under consideration, and $\mathcal{C}_p$ are the positive examples covered by the currently evolved rule.

D. Avoiding overfitting

A serious problem that should be confronted in a data mining task is *data overfitting*. The affinity function described in Section IV-C tends to overfit noisy data, as it favors consistent rules over complete ones. For that reason, an overfitting criterion should be adopted in order to generate accurate rules. The overfitting criterion used in the algorithm presented here is based on the *Minimum Description Length (MDL) principle* [8], [17], which states that shorter rules should be preferred to longer ones [13]. More formally, if H is a set of hypotheses or rules and D is the data, then the MDL principle states:

Minimum Description Length principle: The most preferrable hypothesis from a set of hypotheses H, should be a hypothesis h_{MDL}, such that

$$h_{\mathrm{MDL}} = \underset{h \in H}{\mathrm{argmin}}(L_{C_1} + L_{C_2}),$$

where L denotes length, C_1 is the encoding for the hypotheses set and C_2 is the encoding for the data set.

It is important to state here, that the MDL principle can only provide a clue for the best hypothesis or rule.

Only in case where C_1 and C_2 are optimal encodings for the sets of hypotheses and data, respectively, does the hypothesis h_{MDL} equal the *maximum a posteriori probability* hypothesis, which is the most likely hypothesis of the set H. However, if encodings C_1 and C_2 reflect consistently the possible complexities of hypotheses or data, then the hypothesis h_{MDL} may be a good choice.

A rule in our hybrid technique can be easily and consistently encoded using the already defined GEP antibody encoding. More precisely, the length of a rule h will be the length of its ORF multiplied by the total number of bits needed to encode the different symbols of the GEP alphabet, that is

$$L_h = \log_2(N_c) \cdot L_{\mathrm{ORF}}, \tag{8}$$

where N_c is the total number of symbols, terminal or not, in the alphabet. Therefore, the length of the whole rule set, or the length of the theory L_t, will be the sum of the lengths of all rules in the rule set, so

$$L_t = \log_2(N_c) \sum_i L_{\mathrm{eff}_i}, \tag{9}$$

where L_{eff_i} is the *effective length*, or the length of the ORF, of the i-th rule in the rule set.

In order to consistently and effectively encode the data, we used an approach similar to the one presented in [24]. Only the false classifications are encoded, as the correct ones could be computed from the theory, which is already encoded and transmitted. As well as this, in contrast to the general approach [13], no encoding for the actual class of the missclassification is needed. Indeed, since an one-against-all approach is used for rule generation, we are only interested in whether the rule classifies correctly an example or not. Therefore, the length L_e of the exceptions of a rule can be computed by the formula

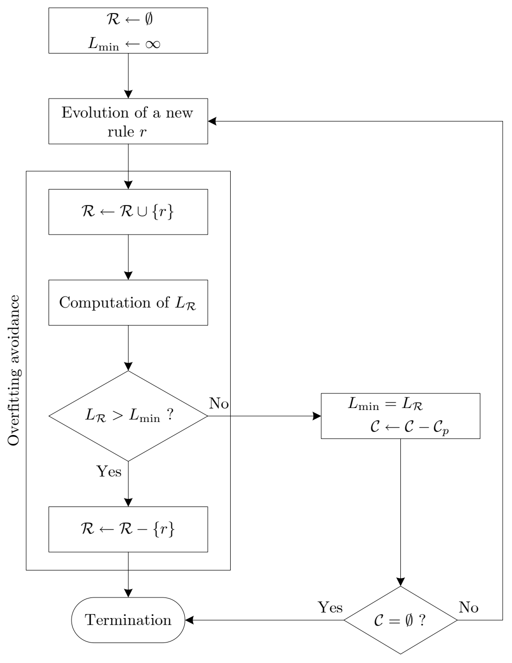
$$L_e = \log_2 \left(\begin{array}{c} N_r \\ N_{f_p} \end{array} \right) + \log_2 \left(\begin{array}{c} N - N_r \\ N_{f_n} \end{array} \right), \tag{10}$$

where N_r is the total number of examples covered by the rule, N_{f_p} is the number of false positives, N_{f_n} is the number of false negatives, and N is the total number of examples. Equation (10) can also be applied to a whole rule set, provided the coverage of a set of rules is defined properly. In our approach, in order to find the coverage of a rule set, all rules in the set are applied sequentially to an example, until one is triggered. In such a case, the example is added to the coverage. If no rule is triggered, then the example is not covered by the rule set.

For the total encoding length of the rule set (theory and exceptions) a weighted sum is used, as in [24], in order to provide more flexibility to the MDL criterion. Specifically, the encoding length $L_{\mathcal{R}}$ of a rule is

$$L_{\mathcal{R}} = L_e + w \cdot L_t, \quad 0 \le w \le 1, \tag{11}$$

where w is the theory weight. If $w = 0$, then theory does not contribute in the total rule set length, which is equivalent to say, that the MDL criterion is not applied at all, as



$\mathcal{R} \leftarrow \emptyset$
$L_{\min} \leftarrow \infty$
Evolution of a new rule r
Overfitting avoidance
$\mathcal{R} \leftarrow \mathcal{R} \cup \{r\}$
Computation of $L_{\mathcal{R}}$
$L_{\mathcal{R}} > L_{\min}$?
No
Yes
$L_{\min} = L_{\mathcal{R}}$
$\mathcal{C} \leftarrow \mathcal{C} - \mathcal{C}_p$
$\mathcal{R} \leftarrow \mathcal{R} - \{r\}$
Termination
Yes
$\mathcal{C} = \emptyset$?
No

Fig. 14. The covering algorithm with the MDL overfitting criterion.

the covering algorithm presented in Section IV-C and in Figure 13 guarantees that the rule set will always cover more examples, therefore leading to less exceptions. In the problems considered in this paper we set $w = 0.1$ or $w = 0.3$ depending on the noise in the data.

The MDL criterion is easily integrated to the covering algorithm already presented by maintaining the least description length $L_{\min}$ encountered so far, and by updating it accordingly at each iteration, as depicted in Figure 14.

E. Generation of final rule set

The final step of the data mining technique presented here is the combination of the independent class-specific rule sets to form a final rule set, which will be able to classify any new example, which will be later presented to the system. Two problems should be coped with in this last part of the process:

- *classification conflict*, where two or more rules classify the same example into different classes, and
- *data rejection*, where an example is not classified at all.

In order to solve the first problem, all produced rules are placed in a single rule set and are sorted according to their affinity—no pruning as in [24] is performed. When a new example is presented, all rules are tried sequentially until one is triggered. The class of the first triggered rule will be the class of the new example. If no rule is triggered, then the problem of data rejection arises, which is solved by defining a default data class.

The default class is defined after the final sorted rule set is formed. All examples of the problem are presented to this rule set and are classified. If an example cannot be classified, then its actual data class is queried and a counter, which counts the unclassified examples of this class, is augmented

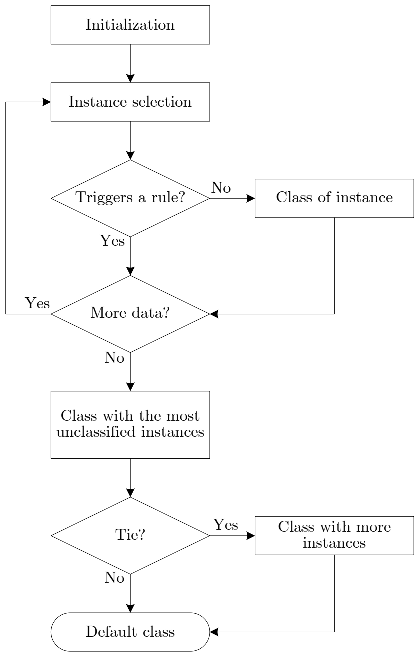


Fig. 15. Algorithm for defining the default data class.

Benchmark Description			
Dataset	*Instances*	*Attributes*	*Classes*
balance-scale	625	4	3
breast-cancer-wisconsin	683	9	2
glass	214	9	7
ionosphere	351	34	2
iris	150	4	3
pima-indians-diabetes	768	8	2
lung-cancer	32	56	3
waveform	5000	21	3
wine	178	14	3

TABLE I

BENCHMARK PROBLEMS USED.

by one. After all examples are presented to the rule set, the class with the most unclassified examples is selected to become the default data class of the problem. Ties are resolved in favor of the class with the most instances. This process is depicted by the flow chart in Figure 15.

V. EXPERIMENTAL RESULTS

The hybrid data mining technique presented so far, was tested against a set of benchmark problems from the UCI repository [14]. Some important information about each benchmark problem is presented in Table I. The purpose of this test was to track the differences in prediction accuracy and in required resources, in terms of convergence rate and population size, between the hybrid technique proposed here and the standard GEP technique proposed in [24].

A. Data preprocessing

In order to evaluate the proposed algorithm, we used a five-fold cross-validation technique. Each dataset was split into five equal subsets. In each run of the algorithm, one such subset was used as the testing set, whilst the remaining four constituted the training set.

However, some preprocessing was necessary for some datasets. More particularly, in the 'breast-cancer-wisconsin' dataset, 16 tuples with missing attributes were eliminated, while, in the 'wine' dataset, some missing attributes were replaced by some random reasonable value. Additionally, the tuples of a number of datasets ('glass', 'iris', 'lung-cancer', and 'wine') were originally ordered per class. In order to better train the algorithm, these datasets were first shuffled and then split into training and testing sets according to the cross-validation technique used. This data shuffling was necessary, so that both the training and the testing set always contained instances of every class of the dataset.

B. Algorithm evaluation

In our previous work [10], we have tested ECA+GEP against the MONK and the 'pima-indians-diabetes' problems and the results were rather satisfactory. In this work, we have tried to tune some algorithm parameters, in order to maximize accuracy and at the same time minimize—as far as possible—the resources used by the algorithm. Although this was not the result of a thorough investigation of every algorithm parameter (see Section VI), we experimentally selected a set of parameter values that could be critical in this tradeoff.

The set of datasets chosen is rather diverse and covers a large spectrum of problem configurations ranging from 2-class up to 7-class problems and from a modest number of 4 attributes to a relatively large description set of 56 attributes in the 'lung-cancer' problem. The algorithm was quite uniformly configured for every benchmark problem, as all of them have numerical attributes. The general configuration is detailed in Table II. Comparing this configuration with the one presented in our earlier work [10], we have increased the population size from 20 to 40 individuals, so as to let diversity emerge more easily. As well as this, antibody length was decreased from 100 to only 40 symbols in total. Although this may signify some loss of expressiveness, it turned out not to be so decisive. On the other hand, the gain in rule clarity, as rules are much shorter now, and the impact of the decrease of antibody length to the execution time of the algorithm, are much more considerable. As a function set for the GEP-antibodies we used a set F of algebraic functions, such that $F = \{+, -, \times, \div, Q, I\}$, where Q is the square root function, and I is the IF function, which is defined as

$$I(x, y, z) = \begin{cases} y, & x > 0 \\ z, & x \leq 0 \end{cases} . \qquad (12)$$

The set of terminal symbols consisted of as many symbols as each problem's attributes plus a set of 4–5 constants, whose values were chosen according to prior knowledge of each benchmark problem's attribute values. Finally, the algorithm was allowed to run for only 50 generations, as the increased population yields better diversity.

Algorithm Configuration	
Parameter	*Value*
Maximum generations	50
Maximum rules/class	3
Gene head length (h)	13
Antibody length (L)	40
Genes/antibody	1
Population size ($\|\mathcal{P}\|$)	50
Memory size ($\|\mathcal{M}\|$)	1
Selected antibodies (n_b)	5
Replaced antibodies (n_r)	0
Refreshed antibodies (n_d)	0
Edited antibodies (n_e)	5
Antibody pool size (n_p)	2
Maximum mutation rate ($\alpha_{\max}$)	1.0
Mutation rate decay (ρ)	5.0
Clone factor (β)	15.0
Theory weight (w)	0.1

TABLE II
GENERAL ALGORITHM CONFIGURATION.

The results of this configuration have well overwhelmed our expectations, as the proposed algorithm has outperformed the standard GEP in almost every benchmark. The results in terms of rule accuracy are summarized in Table III, where a 95% confidence interval is also presented. ECA+GEP achieves better accuracy in every benchmark, except the 'balance-scale' and 'pima-indians-diabetes' datasets, where it is 8% and 1.4% less accurate, respectively. However, the differences in favor of ECA+GEP grow up to about 40% in the 'lung-cancer' dataset and remain above 15% in the 'breast-cancer-wisconsin' and 'waveform' datasets. Finally, another important issue is that ECA+GEP is at least as stable as pure GEP, as it achieves comparable confidence intervals.

However, this increased prediction accuracy does not come at the expense of computational efficiency, as ECA+GEP uses a population of 40 (172 at peak) individuals, which is evolved for only 50 generations. Instead, GEP uses a constant population of 1000 individuals, which is evolved for 1000 generations.

An example ruleset generated by ECA+GEP is presented in Common Lisp notation in the following listing. This rule set is obtained from the 'iris' benchmark and achieves 100% accuracy[3].

```
(cond
  ((> (* (If (* (/ 5 sw) 3)
             (- (+ pl pl) (- 5 1))
             (+ pw pl))
         (- (/ sl 2) (+ sl pl))) 0)
   'Iris-setosa)
  ((> (If pw (/ 5 (- (* pw 3) 5)) sl) 0)
   'Iris-virginica)
  ((> (sqrt (- (/ (sqrt (sqrt 2)) (- 2 pw))
               (sqrt (sqrt (- (+ sw 2)
                              pl))))) 0)
   'Iris-versicolor)
  ((> (+ (- (If 3 (/ (- 2 3) pl) 2)
            (+ (* pw sw) (/ sw 2))) pl) 0)
   'Iris-virginica)
  ((> (- pl (+ 5 (/ 2 3))) 0)
   'Iris-virginica)
  (t 'Iris-versicolor))
```

[3]This accuracy value is achieved on a specific test set obtained after the preprocessing step described and may slightly vary for different subsets of 'iris'.

Rule Accuracy		
Benchmark	*GEP*	*ECA+GEP*
balance-scale	$\mathbf{100.0 \pm 0.0}\%$	$92.0 \pm 3.5\%$
breast-cancer-wisconsin	$76.6 \pm 1.8\%$	$\mathbf{98.2 \pm 0.9}\%$
glass	$63.9 \pm 8.8\%$	$\mathbf{67.5 \pm 12.6}\%$
ionosphere	$90.2 \pm 2.4\%$	$\mathbf{98.3 \pm 2.5}\%$
iris	$95.3 \pm 4.6\%$	$\mathbf{98.3 \pm 1.3}\%$
lung-cancer	$54.4 \pm 15.6\%$	$\mathbf{93.0 \pm 4.7}\%$
pima-indians	$\mathbf{69.7 \pm 3.8}\%$	$68.3 \pm 2.9\%$
waveform	$76.6 \pm 1.4\%$	$\mathbf{93.6 \pm 4.5}\%$
wine	$92.0 \pm 6.0\%$	$\mathbf{97.3 \pm 3.4}\%$

TABLE III
RULE ACCURACY COMPARISON OF GEP AND ECA+GEP.

Execution Times	
Benchmark	*Time (mm:ss.ss)*
balance-scale	04:43.73
breast-cancer-wisconsin	05:36.27
glass	03:22.18
ionosphere	02:30.00
iris	00:56.00
lung-cancer	00:07.88
pima-indians	04:51.61
waveform	44:59.42
wine	01:23.43

TABLE IV
EXECUTION TIMES OF ECA+GEP.

In that listing `If` is a special function defined as in Equation (12) and `sl`, `sw`, `pl`, and `pw`, are the attributes 'sepal length', 'sepal width', 'petal length', and 'petal width', respectively.

C. Resource utilization

In order to obtain a rough estimate of the resources utilized by the proposed algorithm, we have recorded execution times of the algorithm on every benchmark, as well as the maximum resident working set size. The algorithm and the entire framework supporting it was written in the Java programming language (JDK 1.5) and was compiled with Sun's `javac` compiler, version `1.5.0_05`. The algorithm was run on a Pentium 4 machine (2.16Ghz, 1.0GB RAM) running Windows XP Professional SP2. The reported execution time is the total elapsed wall-clock time including initialization times (read input, algorithm setup, etc.), testing times, output dumping, and any overhead incurred by the shell script used to batch-run the algorithm on every benchmark. It is worth noticing that none of the benchmarks needed more than an hour to run, although no special concern for software optimization issues was taken. Execution times range from just about 8sec for small datasets, such as the 'lung-cancer' dataset (32 tuples), and rise up to about 45min for larger ones, such as the 'waveform' dataset (5000 tuples). Despite the fact that we had no direct implementation of GEP in order to comparatively evaluate execution times, we think that ECA+GEP is quite fast for an evolutionary technique. Table IV summarizes execution times for each benchmark problem. As far as memory utilization is concerned, we have not encountered important variation between different benchmarks, as the maximum resident set size depends mainly on the configuration of the algorithm, which was essentially the same for every benchmark examined. More precisely, the process of ECA+GEP consumed a maximum of 131MB of memory during its lifetime.

VI. Conclusions and future work

The immune system is an extremely complex system, which must provide a set of very effective and reliable services and techniques, in order to protect the body from any kind of infection. Modelling such techniques and applying them to machine learning problems is a very challenging task. In this paper, we have modelled the clonal selection mechanism by implementing an extension of the CLONALG algorithm, in order to perform data classification tasks. By coupling this clonal selection model with Gene Expression Programming, we have achieved a considerable reduction in the resources required by the data mining algorithm. Specifically, by applying a set of changes to the conventional CLONALG algorithm, such as addition of a receptor editing step and a more formally defined memory management, we have achieved a considerable amelioration in the convergence rate of the algorithm. Additionally, the proposed algorithm, using a more fine-grained proliferation control, succeeds in maintaining and manipulating a very small initial population, which, even at peak, may become five times less than the population maintained by the conventional GEP technique. By carefully selecting algorithm parameters, a considerable improvement of prediction accuracy compared to conventional GEP may be achieved, while at the same time sparing computational resources.

However, it is obvious that a proper algorithm configuration is essential in obtaining good results. ECA+GEP involves a bunch of parameters that should be tuned, hence a deeper investigation of the algorithm's behaviour against each parameter is a future research prospect. A second step would be to examine different overfitting criteria and how antibodies with multiple genes and their corresponding linking function affect prediction accuracy and convergence rate.

References

[1] F. M. Burnet. *The Clonal selection theory of acquired immunity*. Vanderbilt Univ. Press, Nashville TN, 1959.

[2] D. Dasgupta. Artificial neural networks and artificial immune systems: Similarities and differences, 1997.

[3] D. Dasgupta. *Artifical Immune Systems and their Applications*. Springer Verlag, Berlin, 1998.

[4] L. N de Castro and F. J. Von Zuben. Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6:239–251, June 2002.

[5] C. Ferreira. Gene Expression Programming: A new adaptive algorithm for solving problems. *Complex Systems*, 13(2):87–129, 2001.

[6] C. Ferreira. GEP tutorial. WSC6 tutorial, September 2001.

[7] T. Fukuda, K. Mori, and M Tsukiyama. Immune networks using genetic algorithm for adaptive production scheduling. In *15th IFAC World Congress*, 1993.

[8] P. Grunwald, I. J. Myung, and M. Pitt. *Advances in Minimum Description Length: Theory and Application*. MIT Press, 2005.

[9] J. E. Hunt and D. E. Cooke. Learning using an artificial immune system. *Journal of Network and Computer Applications*, 19:189–212, 1996.

[10] V. K. Karakasis and A. Stafylopatis. Data mining based on gene expression programming and clonal selection. In Gary G. Yen, Simon M. Lucas, Gary Fogel, Graham Kendall, Ralf Salomon, Byoung-Tak Zhang, Carlos A. Coello Coello, and Thomas Philip Runarsson, editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 514–521, Vancouver, BC, Canada, 16-21 July 2006. IEEE Press.

[11] R. P. Lippman. An introduction to computing with neural nets. *Computer Architecture News ACM*, 16(1):7–25, March 1988.

[12] R. Z. Michalski and K. A. Kaufman. A measure of description quality for data mining and its implementation in the AQ18 Learning System. In *International ICSC Symposium on Advances in Intelligent Data Analysis (AIDA)*, June 1999.

[13] T. M. Mitchell. *Machine learning*. McGraw Hill, New York, US, 1996.

[14] D. J. Newman, S. Hettich, C. L. Blake, and C. Z. Merz. UCI repository of machine learning databases, 1998.

[15] G. Nicosia, V. Cutello, and M. Pavone. A hybrid immune algorithm with information gain for the graph coloring problem. In *Genetic and Evolutionary Computation Conference (GECCO-2003) LNCS 2723*, pages 171–182, Chicago, Illinois, USA, 2003.

[16] M. C. Nussenzweig. Immune receptor editing: revise and select. *Cell*, 95(7):875–878, December 1998.

[17] J. Rissanen. MDL Denoising. *IEEE Transactions on Information Theory*, 46(7):2537–2543, 2000.

[18] S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Džeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The MONK's Problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-97, Carnegie Mellon University, December 1991.

[19] F. J. Von Zuben and L. N. De Castro. Artificial Immune Systems: Part I - Basic theory and Applications. Technical Report TR-DCA 01/99, FEEC University Campinas, Campinas, Brazil, December 1999.

[20] F. J. Von Zuben and L. N. De Castro. Artificial Immune Systems: Part II - A Survey of applications. Technical Report TR-DCA 02/00, FEEC University Campinas, Campinas, Brazil, February 2000.

[21] A. Watkins and J. Timmis. Artificial Immune Recognition System (airs): An Immune-Inspired Supervised Learning Algorithm. *Genetic Programming and Evolvable Machines*, 5:291–317, 2004.

[22] R. G. Weinard. Somatic mutation, affinity maturation and antibody repertoire: A computer model. *Journal of Theoretical Biology*, 143:343–382, 1990.

[23] J. A. White and S. M. Garrett. Improved pattern recognition with artificial clonal selection. In *2nd International Conference on Artificial Immune Systems (ICARIS-03)*, pages 181–193, 2003.

[24] C. Zhou, W. Xiao, T. M. Tirpak, and P. C. Nelson. Evolving accurate and compact classification rules with Gene Expression Programming. *IEEE Transactions on Evolutionary Computation*, 7(6):519–531, December 2003.