Expert Systems with Applications 36 (2009) 3996–4005

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Clone selection programming and its application to symbolic regression

Zhaohui Gan, Tommy W.S. Chow *, W.N. Chau

Department of Electric Engineering, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong

ARTICLE INFO

Keywords:
Clone selection
Programming
Immune system
Gene expression

ABSTRACT

A new idea 'clone selection programming (CSP)' is introduced in this paper. The proposed methodology is used for deriving new algorithms in the area of evolutionary computing aimed at solving a wide range of problems. In CSP, antibodies represent candidate solutions, which are encoded according to the structure of antibody. The antibodies are able to keep syntax correct even they are changed with iterations. Also, the clone selection principle is developed as a search strategy. The proposed strategies have been thoroughly evaluated by intensive simulations. The results demonstrate the effectiveness and excellent convergent qualities of the CSP based search strategy. In our study, the convergence rate with respect to population size and other parameters is studied. A thorough comparative study between our proposed CSP based method with the gene expression programming (GEP), and immune programming (IP) are included. The comparative results show that the CSP based method can significantly improve the program performance. The experimental results indicate that the proposed method is very robust under all the investigated cases.

1. Introduction

Evolutionary computing or evolutionary algorithms is a kind of general optimization technology inspired by natural evolution. It has been developed for over three decades since John Holland (Holland, 1975) first introduced genetic algorithms in 1975. Due to its characteristic of independence on the problems, evolutionary algorithms have been widely applied in many physical science and engineering areas, i.e., machine learning, data mining, artificial intelligence, and control systems. In general, evolutionary algorithms can be categorized in four main types, namely, genetic algorithms (GA), genetic programming (GP), evolution strategies (ES) and evolutionary programming (EP) (Eiben & Smith, 2003). Various kinds of evolutionary algorithms differ mainly in the evolution models applied, the evolutionary operators employed, the selection methods and the fitness function used (Fogel, 1994). Genetic algorithms (GA) and genetic programming (GP) are based on the level of genetic. They emphasize the acquisition of genetic structures at symbolic level and regularities of the solutions. On the other hand, the idea of optimization is used in evolution strategies (ES) and the structures being optimized are the individuals of the population. Various behavioral properties of individuals are parameterized and their values evolved with the optimization process. Evolutionary programming (EP) uses the highest level of abstraction by emphasizing the adaptation of behavioral properties of various species (Eiben & Smith, 2003).

Genetic algorithms (GA) are general search methods that use the analogy of natural selection and evolution. Every kind of GA encodes a potential solution of a specific problem in a simple string of alphabets called a chromosome. The basic operators of a GA include individual population initialization, selection, crossover and mutation etc. Through the introduction of nonlinear structures (parse tree) with different sizes and shapes into GA, Cramer introduced Genetic programming (GP) in 1985 (Cramer, 1985) and Koza improved the GP enabling it to be applicable to wider areas, i.e., machine learning, data mining, artificial intelligence, and control systems (Muni, Pal, & Das, 2006). In fact, Genetic programming is a kind of extension of GA (Koza, 1992). The main difference between the GA and GP lies in the representation of the structure and the meaning of representation. GA usually treats a population with fixed or variable-length binary strings or real strings. Parse tree is a kind of structure used for representing computer programs in GP. Since GP focuses on the behavior of computer programs, the definition of phenotype in GP is more abstract than that in GA. In brief, parse tree is a more complex encoding scheme compared to the encoding scheme used in GA, making GP able to create a richer and versatile system representation.

Performing a sequence of functions to the arguments is a kind of simple representation used in most computer programs. Generally, a given computer program is firstly translated into a parse tree by using a language compiler. A sequence of machine instructions is subsequently generated for execution. Thus, parse tree is regarded as natural representations of computer programs at the beginning of GP development (Koza, 1992). In Koza work in GP, a parse tree includes two sets of nodes. The internal nodes are called primitive

* Corresponding author.
E-mail address: eetchow@cityu.edu.hk (T.W.S. Chow).

0957-4174/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2008.02.030

functions, while the leaf nodes are called terminals. The inputs of the program can be represented by the terminals which may include several independent variables and a set of constants. Some primitive functions are combined with the terminals and other primitive functions to form a more complex expression. The above procedures iterate until a desirable program is produced. The selection of the primitive functions set is flexible. It can be freely and randomly generated including different arithmetic operators, logic operators and transcendental functions etc. Generally speaking, there is no fixed guideline on establishing the primitive functions set, because its complexity is mainly dependent upon the given problem.

Apart from basic GP, there are other different improved versions of genetic programming, for example, linear genetic programming (LGP), cartesian genetic programming (CGP) etc. GP on its own lacks a simple and autonomous genome like the linear chromosomes of GA, because the nonlinear structure of parse tree acts as a dual role of genotype/phenotype. Despite chromosomes presenting a kind of structure, GP is still a kind of genetic algorithm because it relies on operating on a population, which consists of numerous parse trees, and selecting a group of improved parse trees according to their fitness. It is worth noting that crossover, mutation, permutation are still the main operators in GP. Although Koza has improved the operations on parse tree by introducing the above three operators, most GP applications use only tree crossover as the genetic operator (Koza, 1992). Consequently, no new genetic material is introduced in the genetic pool of GP populations. Due to the dual role of the parse trees which combining the genotype and phenotype, genetic programming is difficult to incorporate a simple, rudimentary expression. In all cases, the solution must be expressed by using an entire parse tree.

Gene expression programming (GEP) was firstly introduced by Ferreira in 1999 (Ferreira, 2001). GEP incorporates simple fixed length linear chromosomes, which are similar to the ones used in GA. These chromosomes, called genotype in GEP, are changed as the ramified structures of different sizes and shape. In fact, a chromosome can be represented in another form. Phenotype, a ramified expression tree of different sizes and shapes, is translated from a chromosome according to certain criteria. The different formats of genotype and phenotype in GEP are able to bring out the evolutionary advantages of the algorithm.

The phenotype of gene expression programming is the same as the ramified structure used in GP. But the ramified structures evolved by GEP have totally autonomous genome expression. This means that only genome is passed on to the next generation. Consequently, we no longer need to operate the relatively cumbersome structures because all the modifications take place in a simple linear string, which will grow into an expression tree in the later stage. All this novelty comes from the simple, but revolutionary structure of GEP genes. This structure not only allows the encoding of any conceivable program, it also allows an efficient evolution. Although GEP has a powerful encoding scheme, it is still a kind of GA, which possesses all the shortcomings of GA specially the pre-matured.

Artificial immune system is a kind of methodology inspired by the human immune system. Research on artificial immune system (de Castro & Timmis, 2002; Wierzchon, 2002) has become increasingly popular in the area of evolutionary computing. New models of artificial immune system are proposed, and more applied research have been explored, such as computer security, data mining, clustering, pattern recognition and function optimization etc (Cutello, Nicosia, Pavone, & Timmis, 2007; de Castro & Von Zuben, 2000; Watkins, Timmis, & Boggess, 2004). Despite flourishing in some areas, there is few research on immune programming. Musilek, Lau, Reformat, and Wyard-Scott (2006) first proposed a concept, namely, immune programming (IP). IP, inspired by the principles and theories of the immune system, is a novel paradigm composed of the program-like representation of solutions to problems and optimal search engine. IP is an extension of artificial immune algorithms. Clone selection algorithm (de Castro & Von Zuben, 2002, 2000; de Castro & Timmis, 2002), presented by de Castro and Von Zuben, is one of the kinds of artificial immune system (AIS). It has been widely applied into pattern recognition and optimal search. An immune version of GP (iGP) (Nikolaev, Iba, & Slavov, 1999) was introduced by Nikolaev et al. It uses dynamic fitness function implemented by immune network dynamics control the progressive search for programs. The results presented in the paper (Nikolaev et al., 1999) demonstrate that the immune version of GP attains better programs, while it is able to maintain higher population diversity when it is applied to a machine learning task and a time-series prediction problem. Johnson (2003) proposed a kind of artificial immune system programming for symbolic regression. The clone selection algorithm is used as search engine for program.

While computer programs are represented as lisp parse trees like the expression given in the standard GP (Koza, 1992), the results presented in these papers show that these kinds of artificial immune system programming is superior to GP (Johnson, 2003; Musilek et al., 2006; Nikolaev et al., 1999). The current immune programming may be superior to the GP, but the immune programming still has major shortcomings. One of the principal problems is that the algorithm still relies on the use of lisp parse trees or stack representing programs, which result in degrading the efficiency and effectiveness of the algorithm.

In this study, we propose a new clone selection programming to address the aforesaid problems. The main objective of this work is to enhance the effectiveness of programs encoding and search engine. We demonstrate the proposed algorithm by implementing it in the problem of symbolic regression. We analyze search engines from the perspective of immune system theory (Ada & Nossal, 1987; de Castro & Timmis, 2002) and program encoding. The analysis of this type, which has been overlooked in most previous studies, reveals a major drawback of lisp parse trees and stack based encoding. It is found that conventional lisp parse trees and stack based encoding cannot perform optimization in a maximal way because their encoding manner and searching engine do not completely utilize optimization in a way of efficiency and effectiveness. To address this drawback, we employ clone selection programming to formulate a new programming strategy, in which program encoding and search engine appear to be very efficient and effective.

The presentation of this paper is organized as follows. In Section 2, biological immune system is briefed and our proposed programming strategies are detailed. In Section 3, simulation examples are presented and discussed. The conclusion is drawn in Section 4.

2. Clone selection programming

The immune system of human body, which consists of an innate and an adaptive immune system, is a very complex, rapid, and effective defense mechanism against disease. The innate and adaptive immune systems both depend on the activity of a great variety of molecules, cells, and organs spread throughout the body. Various distributed elements incorporate immune functions that do not need central control. The cells of the innate immune system are in-born and exist throughout our body. Once a wide variety of bacteria occur in the body, these cells are immediately available to fight against them. The response produced by an antibody combating a determined infectious agent is called adaptive immune response. The presence of antibodies reflects the kind of infection that our body has already been exposed. Cells of the adaptive

system can extinguish the same antigenic stimulus when these antigens attack the body again. The capability of adaptive immune system is called immune memory which enables diseases within human organism be rapidly destroyed. The adaptive immune system is mainly made up of lymphocytes which are responsible for the recognition and elimination of the pathogenic agents. The main function of the immune system is to protect human organism against pathogens and to eliminate malfunctioning self cells. This self-defense function is largely relied on the ability of recognition. The immune system not only recognizes pathogens and malfunctioning cells, it is also able to recognize the organism's own properly functioning cells and tissues in order to prevent them from inadvertent destruction. All elements, pathogens, malfunctioning cells, and healthy cells etc recognized by the immune system are called antigens. The cells which belong to the organism and are harmless to its functioning are termed self, while the harm-causing elements are termed non-self. The immune system has the ability to distinguish which element is self or non-self. In immunology, this process is named self/non-self discrimination.

When the immune system finds an antigen and recognizes it as non-self, it generates a response to eliminate the pathogen. But the process of antigen recognition and elimination is not enough on their own to deactivate various pathogens. In order to be better to recognize new pathogens and to improve response to pathogens already encountered, the immune system is provided with memory and an ability to learn from the processes of pattern recognition, clone selection, negative selection, and affinity maturation.

In this section, the proposed paradigm of clone selection programming (CSP) is described in detail. In CSP, an antigen is used to present a given problem, while an antibody is used to describe a candidate solution. Each antibody is encoded by a string whose character is analogous to gene in immune systems. The antibody population, randomly generated at initial stage, is evaluated against the problem specification. Their affinities are calculated by mean squared error. If a solution cannot be found in the population, the algorithm will continue to evolve antibodies through replacement, cloning, and hypermutation until a solution is found or a predefined stopping criteria is attained.

2.1. Antibody encoding

2.1.1. Biological structure of antibody

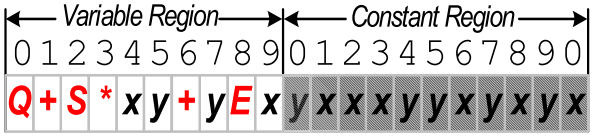
An antibody, or immunoglobulin (Ig) has four polypeptide chains: two identical light (L) and two identical heavy (H) chain (Tonegawa, 1983) composed of an amino terminal region that is highly variable (variable region) and a carboxyterminal region that can be assumed as one of a few types (constant region). The variable region, or *V*-region, is responsible for the antigenic recognition. They contain some special variable sub-regions whose composition is a consequence of the contact with an antigen. Moreover, a number of effector functions, such as complement fixation and ligation to other cell receptors of the immune system, are developed by the constant region, or *C*-region.

Multiple gene segments scattered along a chromosome of a genome consists of a polypeptide chain of an antibody molecule which means that genes located in several different gene libraries are concatenated to form the heavy and light chains of the antibody molecules. For example, the *V*-region of heavy and light chains of the antibody molecules is coded by two separated gene segments named *V* (for variable) and *J* (for junction). Beyond the *V* and J segments, the area between the segments *V* and *J* of the heavy chain is the third segment named *D* which is for diversity. Therefore, antibody population diversity, which is due to the random recombination of gene fragments, is contained in several libraries. The process of somatic hypermutation is used to increase the Ag–Ab (antigen–antibody) affinity so that the immune diversity and capacity of response are improved. The affinity can be understood as the strength of binding between two binding sites, such as a cell receptor and an epitope. Thus, the above two mechanisms of generation and diversification of antibodies makes the immune system capable of synthesizing an almost infinite number of cell receptors from a finite genome.

2.1.2. Structure of antibody encoding

In contrast to its analogous antibody structure expression, antibody in CSP consists of a linear, symbolic string of fixed length that composes of one or more genes. Expression of antibody is rather simple. It has two main components: the antibody encoding and the expression trees. The latter is an expression of the immune information encoded in the former. The process of information decoding (from the antibody encoding to the expression tree) is called translation. The symbols of the antibody and program or problem they represent have a one to one relationship. The spatial organization of the function and terminals in the expression trees is determined by the corresponding rules. Therefore, there exist two languages in CSP: the language of the antibodies and the language of the expression trees. It is noted that the two languages are sufficient to infer exactly the other. For example, the algebraic expression $\sqrt{\sin(xy)+y+e^x}$ can also be used for representing an expression tree (ET), where *Q* represents the square-root function, *E* represents the exponential function, and *S* represents the sinusoidal function. The gene can be represented by the expression tree shown in Fig. 1a.

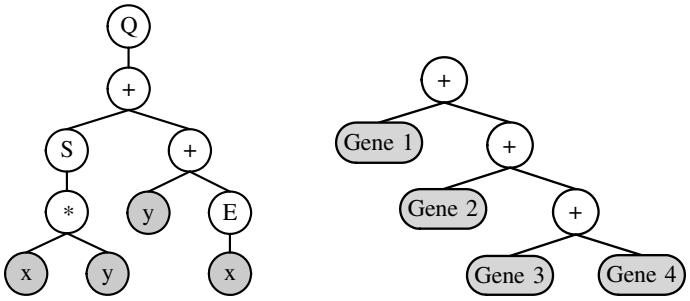
The expression tree as shown in Fig. 1a is, in fact, the phenotype of an antibody gene. An expression string of the above algebraic expression can be translated from the expression tree as follows:



For each sub-tree in the ET, the root must be read out firstly, and then the left child node, the right child node is read out lastly. To complete the translation of the whole expression string to an expression tree, the following rules are used.

(i) The start of expression string corresponds to the root node of the ET as in Fig. 1a.
(ii) If the root node is a terminal, the mapping stops. Or its child branches are processed from left to right. All of the children branches are read from the antibody encoding one by one. If a node is a function having only one argument, only one symbol in the encoding is placed as its child node.If the function has arguments, add a symbol from the left of the start node, some symbols in encoding are placed as its child nodes hierarchically until the end is leaf node. The number of child branches is determined by the number of arguments of their parent node. After the left of node has been constructed, the right part of node will be constructed by the same method.
(iii) From left to right, new nodes are filled consecutively with the elements of the expression string. This process continues in layer-by-layer until all leaf nodes in the ET are composed of elements from the terminal set.

After mapping an antibody encoding into an ET, the fitness of the antibody encoding can be calculated by decoding the ET through traversal. Through various operations, there is no invalid expression or computer program. This encoding scheme makes



(a) The expression tree of a gene (b) A four-gene antibody linked by addition

Fig. 1. The expression tree of gene and antibody.

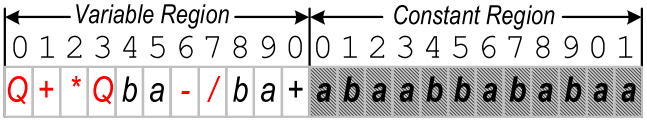
all programs evolved by CSP syntactically correct. Thus, in CSP, syntactic closure is the intrinsic nature making evolution more efficient. Indeed, this is the paramount difference between other immune programming and other types of GP implementations, which either limit themselves to inefficient genetic operators or checking all the newly created programs exhaustively for syntactic errors.

2.1.3. Structure and functional organization of antibody encoding

The encoding scheme is similar to gene expression encoding in GEP (Ferreira, 2006). The structure of antibody genes composes of two different domains which exhibit different properties and functions. They are a variable region and a constant region. The variable region is used mainly to express the functions chosen for the specific problem, whereas the constant region works as a buffer or reservoir of terminals in order to guarantee the formation of only valid structures. Thus, the variable region contains symbols representing both functions and terminals, whereas the constant region composes of only terminals. For a given problem, the length of the variable region V is chosen, whereas the length of the constant region C is a function of V and the number of arguments of the function with more arguments $n_{\max}$ (also called maximum arity). It is evaluated by

$$C = V * (n - 1) + 1. \tag{1}$$

Consider a gene, a set of functions $F = \{Q, *, /, -, +\}$, and a set of terminals $T = \{a, b\}$ are given. Thus, it gives $n_{\max} = 2$, and if we chose an $V = 11$, then $C = 11 \cdot (2 - 1) + 1 = 12$, the length of the gene is 11 + 12 = 23. A typical gene is shown below, in which the constant region is shown in bold



The antibodies usually consist of several genes of equal length. The interaction between the genes is specified by a linking function. An example of a four-gene antibody linked by addition is shown in Fig. 1b.

2.2. Clone selection algorithm

Based on the concept of the clone selection in immune system (Nossal, 1993), the algorithm of CSP is developed. At the initialization stage, antibodies, which are the candidate solutions to a given problem, are randomly generated as an initial population. The cloning and hypermutation operation of the antibody make the population evolve so that the diversity of the population is maintained and the search space for solutions is expanded. Evaluation of the quality of each antibody is based on the affinity value. The high-affinity antibodies are selected for cloning or hypermutation, while the antibodies with lower affinity are replaced. This selection process finishes through one of the operations of replacement, cloning or mutation to form a new population. This process repeats until the stopping criteria or maximal generation number is reached. The final result of search process is then decoded to the program space becoming an implementation of the solution to the given problem.

The procedures of CSP can be briefly described as follows:

A. *Initialization*: The initial population of antibodies is randomly generated. The encoding structure is introduced in the above section. These N individuals compose of an initial population (P) of candidate solutions which include a subset of memory cell (M).

B. *Evaluation*: An antigen (Ag) representing the problem to be solved appears at the initial stage of programming. Different expression forms of antigen depend on the specific problem. Here, we assume that the antigen is taken the form of an arithmetic expression, say, $\mathrm{Ag} = x^2 + y^2 + x + y$. To evaluate the affinity of the antibodies, particular values of variable(s) have to be placed on the expression and the programs have to be executed. Since the problem is described in a symbolic form, no numerical argument values are explicitly prescribed and they must be generated. For the antigen used as an example, this corresponds to generation of x and y values. Here, we randomly generate x and y in the range of [0,255]. Five sets of values x and y are generated to execute each antibody and to compare the execution results to the antigens behavior. The affinity of ith antibody with antigen $f_i = \frac{1}{\frac{1}{n}\sum_{i=1}^{n}(\mathrm{Ab}_i - \mathrm{Ag}_i)^2 + 1}$, where $\mathrm{Ag}_i = x_i^2 + y_i^2 + x_i + y_i$. $\mathrm{Ab}_i = f(x_i, y_i) \cdot n = 5$. Thus, the affinity of antibody with antigen is in the range of [0,1]. The whole antibodies are sorted in descending order according to the affinity of all antibodies in the population P with antigen.

C. *Cloning*: Before a new antibody has been generated, an antibody Ab_i from the current population is considered for cloning. The antibodies are selected for cloning according to the affinities with antigen. The antibodies in population are sorted in descending order. The n ($n < N$) highest affinity antibodies will be cloned. The number of clones (given by Eq. (2)) reproduced for each individual is proportional to its affinity.

$$N_c = \mathrm{round}\left(\frac{\beta * N}{i}\right) \tag{2}$$

for each individual, β is a multiplying factor. Where, N_c is the number of clones generated, N is the total number of individuals, i is the index of current individual in the population, and round $(\cdot)$ is the function that rounds its variable toward the closest integer. After these n best individuals are cloned, a temporary clone population (P_C) is generated.

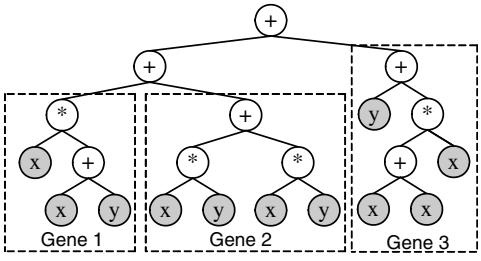
D. *Hypermutation*: The individuals in the population P_C of the previous step are submitted to a hypermutation procedure. Suppose an antibody has J genes, while a gene has K bit symbol string, $Ab_i = \langle S_1, S_2, \ldots S_j \rangle$, the mutation process is implemented by replacing some bit symbols of each gene with some new randomly generated symbols belonging to the defined function set or terminal set. The symbols in variable region may be replaced as function or terminal symbol, whereas the symbols in constant region are only mutated as terminal symbol. The probability of mutation P_m determines this process. To illustrate the process of hypermutation, let us assume that there is an antibody containing three genes, the size of variable region of gene is five, the argument number is two, and the size of constant region of gene is six. This antibody is shown as follow in which the constant regions are shown in bold:

```
01234567890012345678900123456789 0
*x+xyyxxxyx+**xyxyxyxx+y*+xxxyxyy
```

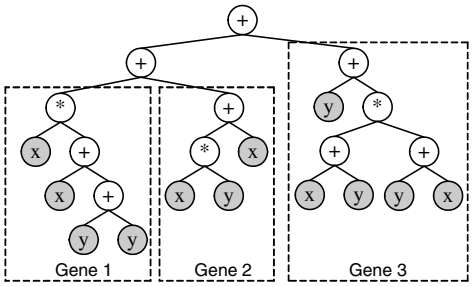
The corresponding arithmetic expression is $f(x,y) = 4x^2 + 2xy + y$, and its corresponding expression tree is shown in Fig. 2a. From the first symbol of antibody, a random positive float number less than one is generated. If the number is less than the probability of mutation, P_m, the symbol will mutate. The new symbol that replaces the old one is randomly selected from the function or terminal set. This process repeats until the last symbol of the last gene. The Pseudocode of mutation is shown in Fig. 3. In our example, mutation points of each gene of antibody are highlighted in underline. The mutated antibody is shown as follows:

```
01234567890012345678900123456789 0
*x+x+yyxxyx+*xxyxyxyxx+y*++xyyxxy
```

The corresponding arithmetic expression is $f(x,y) = 3x^2 + 4xy + 2y$, and the corresponding expression tree is shown in Fig. 2b. Owing to the hypermutation operator, the algorithm is able to provide new gene material into the antibody population. This can enhance the diversity of antibody population and expand the search space for finding the solution. After hypermutation, an antibody population (P_M) based on clone population (P_C) is generated.



(a) The expression tree of antibody



(b) The expression tree of antibody mutated

Fig. 2. The expression tree of antibody and antibody mutated.

```
Mutation
   for each gene g(g=1,2,3,…,N) in the antibody
       for each symbol i(i=1,2,3,…,L) in the gene
         if rand(1) < Pm
            if i < head_length
          replace symbol i by a randomly-generated
          function or terminal
            else
               replace symbol i by a randomly-
generated or terminal
     end
```

Fig. 3. Pseudocode of mutation.

```
Input: Ag,MaxGen,N,n,βd,r
Output: the best antibody in memory cell set M
Initialize the population P (N antibodies AbN)
for t = 1 to MaxGen
  f = Evaluate(AbN,Ag);// evaluate all antibodies in
population P
  Abn = Select(f);//select n(n<N) antibodies with higher
affinity
  AbC = Clone(Abn,β);     //generate clone population PC
  AbM =  Mutation(AbC);  //generate mutated clone
                         population PM based on clone
                         population PC
  f* = Evaluate(AbM,Ag);//re-evaluate antibodies in PM
    AbN =  Reselect(AbM,f*, AbN);//re-select r best
                           antibodies becoming  memory
                         cell set M and replace r worst
                           antibodies in population P
                                    with M
  Abd =  Generate(d);     //generate d new random
```

Fig. 4. Pseudocode of whole algorithm.

E. *Re-selection*: After hypermutation is finished, the individuals in the population P_M are evaluated again so that the individuals with higher affinity are chosen to compose the memory cell set M.

F. *Replacement*: After the above phases are complete, the algorithm proceeds with generation of new individuals. The new randomly generated individuals will be put into population directly so that the lower affinity individuals will be replaced with higher probabilities.

Step (ii) to step (vi) iteratively proceed until the stopping criterion is reached. The criterion used in this study takes two forms: maximal number of generations and fitness threshold. At last, the final attribute string is presented and translated into solution of specified problem. The Pseudocode of the whole algorithm is depicted in Fig. 4.

3. Simulations and analysis

To demonstrate the advantage of our proposed algorithm, several simulations with antigens in the form of arithmetic expressions are conducted. The results are compared with mainly related methods such as immune programming (IP) and gene expression programming (GEP). The function and terminal set are $F = \{+, *, /, -\}$, and the set of terminals are $T = \{x, y\}$.

Besides the form of the generated programs, the number of generations to arrive at the solutions is recorded. Due to the stochastic nature of the algorithm and simplicity of arithmetic expressions, the foregoing four examples were conducted with 1000 independent trials to avoid bias. The results are averaged and reported in

Table 1
The parameters of CSP

Population size, N	100
Number of antibodies selected, n	20
Clone factor, β	2
Number of new antibodies, d	40
Number of antibodies replaced, r	80
Mutation rate, P_m	0.2

Table 2
The parameters of GEP

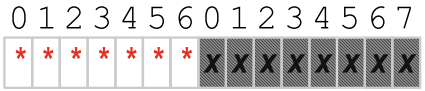
Population size, N	100
Mutation rate	0.44
Inversion rate	0.1
IS transposition rate	0.1
RIS transposition rate	0.1
Gene transposition rate	0.1
One-point recombination rate	0.3
Two-point recombination rate	0.3
Gene recombination rate	0.3

this paper. As stacked encoding used in IP is totally different to antibody encoding in CSP and gene expression encoding in GEP, a program size expressed by stack encoding, antibody encoding and gene expression encoding is set as the same for fair comparison. The parameters used in CSP and GEP are listed in Tables 1 and 2. The results of the simulations of IP are abstracted from this paper (Musilek et al., 2006).

3.1. Single variable high order expression

This simulation is used to evaluate the performance of three algorithms handling with operations of high order single variable expression. The expression used is X^8. The performance of three algorithms over the program size, L, is listed in Table 3. As the minimum number of antibody encoding and gene expression encoding for this expression is six, CSP and GEP do not deliver results when the program size is five.

In IP, the numbers of generations to find a solution are all inverse proportional to the program length, whereas in GEP and CSP, the numbers of generations are nearly constant due to this simple expression. Compared with IP and GEP, our proposed approach requires less number of generations to obtain a solution in all simulations with different program sizes. The shortest possible program is in the following form corresponds to the expression X^8.



3.2. Multiple variable expression

In this session, we demonstrate the ability of three algorithms to handle operations of multiple variable mathematical expression.

Table 3
Comparison of performance of IP, GEP and CSP for high order operations

Constrained program size L/variable region size H	6	7	8	9	10
Number of generation G (IP)	318.3	58.3	55.2	54.6	33.3
Number of generation G (GEP)	N/A	4.8	4.7	6.3	7.4
Number of generation G (CSP)	N/A	2.6	2.3	2.5	2.7

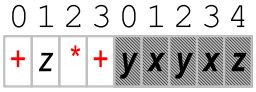
The number of generations is an average of a number of different trails.

Table 4
Comparison of performance of IP, GEP and CSP for multiple variables

Constrained program size L/variable region size H	4	5	6	7	8	9	10
Number of generation G (IP)	10.7	14.6	10.9	17.1	15.9	26.8	43.4
Number of generation G (GEP)	23.5	20.7	42.8	85.9	130.9	158.1	195.1
Number of generation G (CSP)	11.7	6.2	9.7	14.2	17.6	21.6	24.3

The number of generations is an average of a number of different trails.

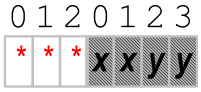
The particular expression used in this study is $x * y + y^2 + z$. It involves three variables x, y, z. The performance of the three algorithms for various values of L, is shown in Table 4. It shows that stack based encoding scheme is unable to represent a relatively complex algebraic structure. Compared with IP and GEP, our approach requires less number of generations to come up with the solution in all simulations with different program sizes. The shortest possible program is in the form of



The above expression corresponds to a mathematical expression of $x * y + y^2 + z$.

3.3. Factorization

In order to compare the ability of the three algorithms for simplifying an arithmetic expression by factorization, this session demonstrates an antigen expression of $x^2 * y^2$. The performance of the three algorithms for various values of L, is summarized in Table 5. For the IP, the obtained results show that there is no strong dependence between the number of generations and the program length, whereas the number of generations required by our proposed approach is more or less constant. Compared with the IP and GEP, our proposed approach requires less number of generations to find a solution. The shortest possible program is in the form of



The above expression corresponds to a mathematical expression of $x^2 * y^2$.

Table 5
Comparison of performance of IP, GEP and CSP for factorization

Constrained program size L/variable region size H	3	4	5	6	7	8	9	10
Number of generation G (IP)	4.8	4.7	4.5	6.4	6.3	11.4	9.2	12
Number of generation G (GEP)	2.75	2.78	4.44	12.91	32.58	39.60	47.77	53.53
Number of generation G (CSP)	1.71	1.50	1.61	2.75	4.96	5.50	5.83	6.35

The number of generations is an average of a number of different trails.

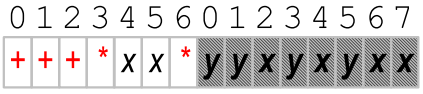
Table 6
Comparison of performance of IP, GEP and CSP for no simplification

Constrained program size L/head length H	7	8	9	10	11	12	13	14	15
Number of generation G (IP)	N/A	N/A	N/A	488.1	292.3	210.4	165.2	135.3	224
Number of generation G (GEP)	21.8	36.9	60.9	117.0	204.4	249.3	309.9	293.7	326.6
Number of generation G (CSP)	6.5	9.9	17.3	28.8	54.3	55.9	51.6	54.7	53.2

The number of generations is an average of a number of different trails.

3.4. No simplification expression

In order to verify whether the three algorithms are capable of solving more complicated mathematical expression. In this session, an expression $x^2 + y^2 + x + y$, which cannot be further mathematically simplified, is considered. The performance of the three algorithms over various values of L, is summarized in Table 6. It is noticed that a minimum of 10 instructions is required for IP to solve the expression. Apparently, the increased program length causes a significant increase in the number of generations. In our proposed algorithm, a minimum length of 7 variable regions is required to represent the expression. It is worth noting that they are all able to obtain the correct solutions irrespective of minimum or maximum length of gene expression encoding or antibody encoding. It is important to point out that our proposed approach has exhibited faster convergence rate compared with IP and GEP. The shortest possible program is in the form of



The above expression corresponds to the mathematical expression of $x^2 + y^2 + x + y$. The evolution of affinity and the sensitivity of the algorithm parameters are also examined. A comparative study of the three algorithms is detailed in Section 3.5.

3.5. Evolution of affinity

The same expression $x^2 + y^2 + x + y$ is used in this analysis. The aim of this session is to analyze the evolution of population with respect to its affinity. The parameters of CSP are set to $P = 100$, $n = 20$, $d = 40$, $r = 80$, $\beta = 2$, $P_m = 0.2$, where n, d, r, β, P_m are chosen according to the empirical results from the sensitivity analysis in the following section. The program size is set to $L = 6$, the evolved function set is $\{+, -, *, /\}$, terminal set is $\{x, y\}$. In the trial under examination, the algorithm found the solution in the 18th generation. The growth of affinity during evolution of the solution is shown in Fig. 5. The maximum and average affinities are gradually improved until the 15th generation at which a marked improvement on the maximum and average affinities appears. The maximum and average affinities are obtained at the 18th generation when the solution is correctly found.

3.6. Sensitivity analysis

To examine the parameter sensitivity of our proposed algorithm, the same expression $x^2 + y^2 + x + y$ as used in Section 3.5 is considered. The following parameters with their default values indicated in parentheses will be changed.

Population size N ($N = 100$),

Percentage of antibodies selected for clone to population n ($n = 0.2$),

Clone factor β ($\beta = 2$),

Percentage of new antibodies to population d ($d = 0.4$),

Percentage of antibodies replaced to population r ($r = 0.8$),

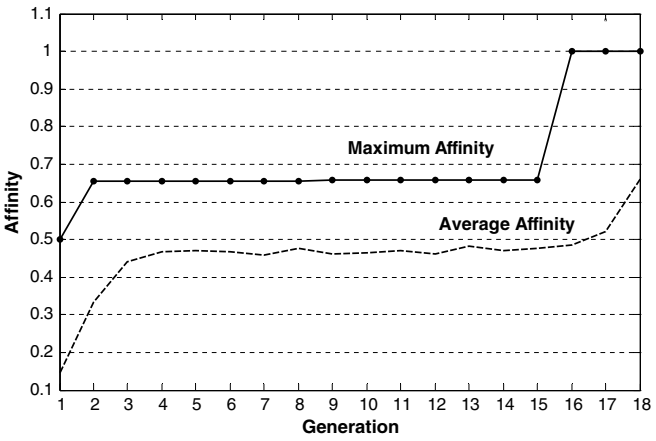


Fig. 5. Evolution of affinity over time.

Probability of mutation P_m ($P_m = 0.5$).

In these experiments, the variable region size of antibody encoding, L, is kept constant at 10. Similar to previous examples, the results of 100 independent trials are averaged to avoid bias.

3.6.1. Effect of population size

This session examines the effect of population size on convergence rate. The population was varied from 20 to 1000 with an increment of 10. The number of generations required for producing the solution is plotted against the population size shown in Fig. 6. The curve indicates that a large population size would have an effect on speeding up the convergence rate. But it is noticed that the algorithm was able to converge and to find a solution even with a very small population size. The obtained result also indicates that there is a threshold on the population size, which appears to be about 300, beyond which has no further effect on speeding up the convergence rate.

3.6.2. Effect of percentage of antibodies selected for clone to population

We examine the effect of varying the percentage of antibodies selected for clone to population, n, on the convergence. In this

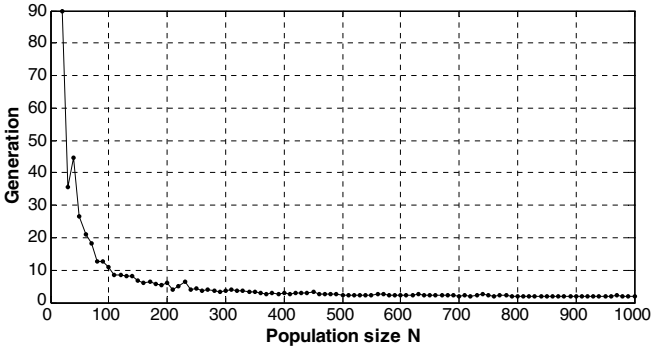


Fig. 6. Sensitivity of CSP performance with respect to population size, N.

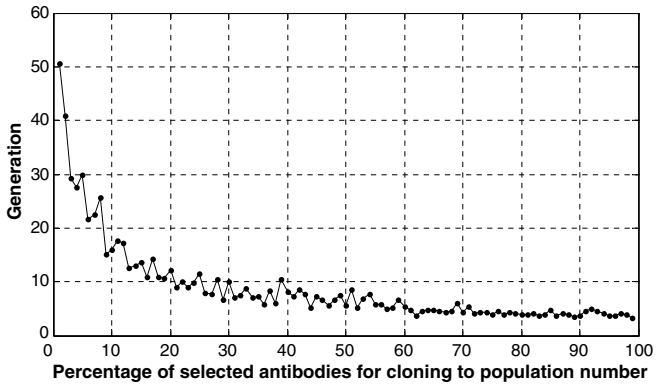


Fig. 7. Sensitivity of CSP performance with respect to size of selected antibody for clone, *n*.

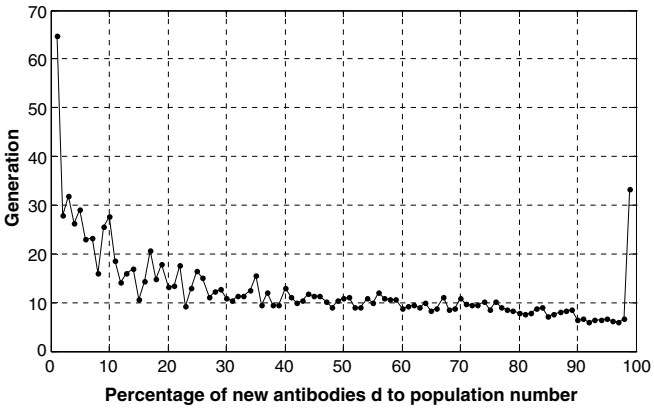


Fig. 9. Sensitivity of CSP performance with respect to size of new antibodies, *d*.

study, *n* was varied from 1% to 99% with an increment of 1%. The number of generations required to produce the solution is plotted against *n*. Fig. 7 shows that a larger number of antibodies selected for clone has a significant effect on speeding up the convergence rate. The low convergence rate at a small *n* is attributed to the decrease in search space.

3.6.3. Effect of clone factor

This session evaluates the effect of clone factor on convergence rate. The clone factor β was varied from 0.1 to 10 with an increment of 0.1. Fig. 8 shows the number of generations required to produce the solution is plotted against the clone factor, β. The curve shows that a larger number of clone factor has a significant effect on speeding up the convergence rate. It is noticed that a small value of β will have an effect of slowing down the convergence rate. This is mainly due to a loss of search space.

3.6.4. Effect of percentage of new antibodies to population

We examine the effect of varying the percentage of new antibodies to population, *d*, on the convergence rate. In this study, *d* was varied from 1% to 99% with an increment of 1%. The number of generations required to produce the solution is plotted against *d* in Fig. 9. It shows that a larger number of new antibodies has a significant effect on speeding up the convergence rate. The low convergence rate at a small *d* is attributed to the decrease in search space.

3.6.5. Effect of percentage of antibodies replaced to population

In this session, we evaluate the effect of percentage of antibodies replaced to population on the convergence rate. The percentage of antibodies replaced, *r*, was varied from 1% to 99% with an increment of 1%. The number of generations required to produce the solution is plotted against *r*. Fig. 10 shows that the percentage of antibodies replaced, *r*, do not affect the convergence rate significantly.

3.6.6. Effect of probability of mutation

We evaluate the effect of probability of mutation, P_m, on the convergence rate. The value of P_m was varied from 1% to 90% with an increment of 5%. The number of generations required to produce the solution is plotted against the probability of mutation, P_m. In Fig. 11, a "*V*" shape with a minimum at 0.5 appears in the curve. It is noted that the probability of mutation significantly affects the efficiency of our proposed algorithm.

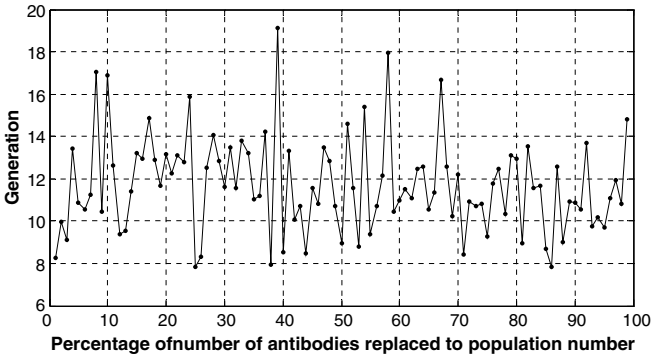


Fig. 10. Sensitivity of CSP performance with respect to size of replaced antibodies, *r*.

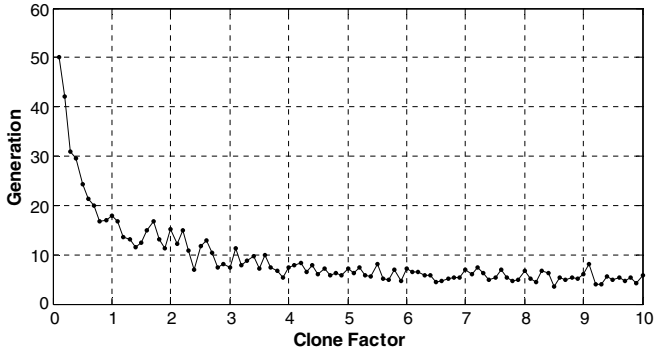


Fig. 8. Sensitivity of CSP performance with respect to clone factor, β.

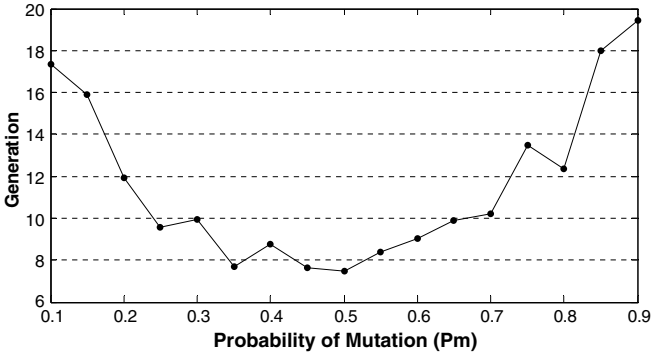


Fig. 11. Sensitivity of CSP performance with respect to probability of mutation, P_m.

3.7. Comparison of between CSP, IP and GEP

In evolutionary computation, encoding and optimal search strategy are the most important aspects affecting the performance of the algorithms. GA usually requires large numbers of encoding schemes and search strategies. In evolutionary programming, Gene expression programming and immune programming are very versatile algorithms. In the following section, we thoroughly analyze and compare their advantages and shortcomings from the perspectives of encoding scheme and search strategy.

3.7.1. Encoding scheme

Stack-based encoding scheme has the characteristics of small size, and low system complexity. It is able to deliver high system performance consistently even under varying conditions (Jerne, 1974). But it has the following distinct shortcomings:

- It is difficult to express complex program.
- Efficiency of executing program is low.
- It cannot guarantee program syntax correct.

In antibody encoding scheme, it has an important characteristic of being syntactic closure. Antibody encoding scheme allows the genotype totally unconstrained manipulated, which results in an efficient evolution. This is the paramount difference between antibody encoding and stack encoding or previous lisp tree encoding in GP implementations, the latter two encoding scheme make some evolution operation quite complex and inefficient because algorithm must check exhaustively all the newly created programs for syntactic errors (Banzhaf et al., 1998).

3.7.2. Search strategy

In fact, gene expression programming is a special genetic algorithm whose encoding represents computer program. It has all of genetic operators of genetic algorithm and increases some new genetic operators. Although GEP outperforms GP due to its powerful gene expression encoding, its search strategy basically is the same as genetic algorithm, which it has all shortcoming of genetic algorithm. Clone selection algorithm based on biological immune system is a kind of new optimal search approach, which has strong search ability. Although CSP and IP all are clone selection algorithm, there are some difference between them listed below.

- In CSP, the number of new generated individual in each generation keeps constant, whereas in IP, it depends on the probability of clone Pc and affinities of individuals.
- In CSP, only some high-affinity individuals will be chosen for clone, whereas low-affinity individuals do not have chance to be cloned.In IP, the more larger affinity individuals have, the higher chance individuals have to be cloned.
- In CSP, only cloned individuals can be mutated, which these individuals have equal chance be mutated.In IP, each individual not cloned has chance to be mutated, which probability of mutation depends on their affinities.
- In CSP, the individuals being cloned and mutated are the best in population.In IP, although clone and mutation of individuals depend on their affinities, the highest affinity individuals are probably not be evolved.
- In CSP, Due to syntactic closure of antibody encoding, we just need simple Euclidean distance to calculate affinity of the generated programs from the expected results defined by the antigen.In IP, a complex evaluation function must be elaborately designed for calculating affinity of individual because stack-based encoding does not guarantee all program syntax correctness, if evaluation function is not be designed better, it will decrease the performance of algorithm.

In order to further compare the performance of CSP with IP and GEP, the expression $x^2 + x + 3y$ is used for this study. To allow an unbiased comparison, the same method of generated test cases and evaluation of candidate solutions were maintained. As the minimal program size for this expression of these two encoding schemes is five and nine respectively, the simulations are conducted for the minimal program length, $L = 5$ and $L = 9$, and another larger size, $L = 15$. The stopping criteria is either success in finding a solution or a maximum number of generations, $G_{max} = 2000$. Each of the experiments is repeated one hundred times. The results, listed in Table 7, represent the average taken over all trials. The table also relates the number of trials (as a percentage of the one hundred trials) in which a solution is found within G_{max} generations. This measure is provided since in some trials neither algorithm finds a solution within the stopping criteria. The average number of generations G to find a solution for 100% successful sets of trials is set in a bold font. For any population size and any program size, clone selection programming is clearly superior to IP and GEP not only in terms of the average number of generations needed to find a solution, but also in the ability of the algorithm to find a solution within a restricted number of iterations. When program size is minimum number for solving this expression in IP, IP gets quite low success rate in all trials for any population of size smaller than 1000. The CSP and GEP algorithm, on the other hand, achieve complete success for any population of size, but CSP runs faster than GEP. When program size is set as fifteen, the CSP algorithm is able to find a solution in 95% of trials for population size $N = 10$. The ability of CSP to perform successfully with small populations can be attributed to the way in which the algorithm has better encoding and powerful search strategy which maintains population diversity. This diversity is introduced during initialization steps which are analogous to population initialization of other two algorithms. However, in CSP, diversity is subsequently maintained via replacement and mutation. Additionally, due to the affinity-based selection process, the high-affinity individuals will be cloned and kept to next generation and mutation happens at individuals having been cloned so that the most excellent individuals were chosen and evolved.

4. Conclusion

In this paper, a new paradigm of evolutionary computation named 'clone selection programming' (CSP) is introduced. Based

Table 7
Comparison of convergence success rate of IP, GEP and CSP

L/H	System	Repertoire/population size, n							
		10	50	100	200	300	400	500	1000
5	IP	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
		0%	0%	0%	0%	0%	0%	0%	0%
	GEP	135	37	29	18	11	11	9	3
		100%	100%	100%	100%	100%	100%	100%	100%
	CSP	157	23	17	5	3	3	3	2
		100%	100%	100%	100%	100%	100%	100%	100%
9	IP	N/A	919	1053	991	852	846	857	786
		0%	7%	12%	24%	35%	43%	51%	75%
	GEP	148	50	33	29	23	15	13	6
		99%	100%	100%	100%	100%	100%	100%	100%
	CSP	63	10	5	3	3	2	2	2
		100%	100%	100%	100%	100%	100%	100%	100%
15	IP	1057	1001	807	795	650	529	436	249
		10%	30%	53%	82%	90%	99%	99%	100%
	GEP	511	221	181	167	161	134	101	100
		85%	98%	100%	100%	100%	100%	100%	100%
	CSP	188	26	17	6	5	4	4	3
		95%	100%	100%	100%	100%	100%	100%	100%

on the biological immune system concepts, CSP is an extension of artificial immune system (AIS), which is a systematic, domain-independent, and intelligent based method to solve programming problems. A specific operation is implemented by an antibody's affinity and a set of probabilistic parameters. Convergence of CSP is superior to IP and GEP for the problems tested: successful solutions are found in fewer generations with the most dramatic improvement evident when using a small antibody population. Additionally, CSP converges in situations that cannot be handled by GEP and IP, which is attributed to its powerful encoding scheme and search strategy. Sensitivity of CSPs convergence with respect to algorithm parameters is explored in this paper. The generalization capabilities of the system are demonstrated by its capacity to provide a variety of alternative solutions to a given problem.

It is worth poting out that the paradigm may be widely applied into machine learning, data mining and optimization, i.e., classification of datas, feature selection and system recognition and modelling etc. In future work, we will extend this algorithm to new applications in order to further evaluate its advantages and improve its performance.

References

Ada, G. L., & Nossal, G. J. V. (1987). The clonal selection theory. *Scientific American, 257*(2), 50–57.

Banzhaf, W., Nordin, P., Keller, R. E., & Francone, F. D. (1998). *Genetic programming: An introduction: On the automatic evolution of computer programs and its applications*, Morgan Kaufmann.

Cramer, N. L. (1985). A presentation for the adaptive generation of simple sequential programs. In *Proceedings of the international conference on genetic algorithms and their applications* (pp. 183–187).

Cutello, V., Nicosia, G., Pavone, M., & Timmis, J. (2007). An immune algorithm for protein structure prediction on lattice models. *IEEE Transactions on Evolutionary Computation, 11*(1), 101–117.

de Castro, L. N., Von Zuben, F. J. (2000). The clonal selection algorithm with engineering applications. In *Proceedings of GECCO'00, workshop on artificial immune systems and their applications* (pp. 36–37).

de Castro, L. N., & Timmis, J. (2002). *Artificial immune systems: A new computational intelligence approach*. Springer.

de Castro, L. N., & Von Zuben, F. J. (2002). Learning and optimization using the clonal selection principle. *IEEE Transaction on Evolutionary Computation, Special Issue on Artificial Immune Systems, 6*(3), 239–251.

Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing*. Springer.

Ferreira, C. (2001). Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems, 13*(2), 87–129.

Ferreira, C. (2006). *Gene expression programming: Mathematical modeling by an artificial intelligence*. Springer.

Fogel, D. B. (1994). An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks, 5*(1), 3–14.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press.

Jerne, N. K. (1974). Towards a network theory of the immune system. *Annales D Immunologie (Inst. Pasteur), 125C*, 373–389.

Johnson, C. G. (2003). Artificial immune system programming for symbolic regression. In *Genetic programming, 6th European conference (EuroGP)* (pp. 345–353).

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press.

Muni, D. P., Pal, N. R., & Das, J. (2006). Genetic programming for simultaneous feature selection and classifier design. *IEEE Transactions on Systems, Man and Cybernetics, Part B, 36*(1), 106–117.

Musilek, P., Lau, A., Reformat, M., & Wyard-Scott, L. (2006). Immune programming. *Information Sciences, 176*(8), 972–1002.

Nikolaev, N. I., Iba, H., & Slavov, V. (1999). *Inductive genetic programming with immune network dynamics. Advances in genetic programming* (Vol. 3). MIT Press (pp. 355–376).

Nossal, G. J. V. (1993). The molecular and cellular basis of affinity maturation in the antibody response. *Cell, 68*, 1–2.

Tonegawa, S. (1983). Somatic generation of antibody diversity. *Nature, 302*, 575–581.

Watkins, A., Timmis, J., & Boggess, L. (2004). Artificial immune recognition system (AIRS): An immune inspired supervised machine learning algorithm. *Genetic Programming and Evolvable Machines, 5*(3), 291–317.

Wierzchon, S. T. (2002). Function optimization by the immune metaphor. *Task Quarterly, 6*(3), 493–508.