A novel algorithm of artificial immune system for high-dimensional function numerical optimization*

DU Haifeng[1,2**], GONG Maoguo[1], JIAO Licheng[1] and LIU Ruochen[1]

(1. Institute of Intelligent Information Processing and Key Laboratory of Radar Signal Processing, Xidian University, Xi'an 710071, China; 2. School of Mechanical Engineering, Xi'an Jiaotong University, Xi'an 710049, China)

Received August 15, 2004; revised September 28, 2004

Abstract Based on the clonal selection theory and immune memory theory, a novel artificial immune system algorithm, immune memory clonal programming algorithm (IMCPA), is put forward. Using the theorem of Markov chain, it is proved that IMCPA is convergent. Compared with some other evolutionary programming algorithms (like Breeder genetic algorithm), IMCPA is shown to be an evolutionary strategy capable of solving complex machine learning tasks, like high-dimensional function optimization, which maintains the diversity of the population and avoids prematurity to some extent, and has a higher convergence speed.

Keywords: clonal selection, immune memory, artificial immune system, evolutionary algorithms, Markov chain.

Artificial immune system has become a research hot spot after the neural network, fuzzy logic and evolutionary computation[1,2]. Clone means propagating asexually so that a group of genetically identical cells can be descended from a single common ancestor, such as a bacterial colony whose members arise from a single original cell as the result of binary fission. The idea has been extensively applied in some fields like computer programming[3,4], system control[5], interactive parallel simulation[6] and so on. In artificial immune system, clonal mechanism attracts great attention from the artificial intelligence researchers[7,8]. However, clonal algorithms based on the antigen clonal mechanism are very few, and most of algorithms developed have not considered entitative features of clonal selection process such as memory and chaos dynamics, and even no complete analysis (like convergence) has been accomplished.

In this paper, based on the antibody clonal selection theory and immune memory theory, a novel artificial immune system algorithm, immune memory clonal programming algorithm (IMCPA), is put forward. Different from other clonal selection algorithms, the new algorithm adopts decimal coding instead of binary coding, and intends to combine manifold dominations of antibodies and antigens into an antibody in order to embody the cooperation among antibodies and increase the diversity of the population. The new algorithm can accelerate affinity maturation speed by constructing a memory cell. The results of theoretical analysis and the experiments indicate that when compared with some evolutionary algorithms such as BGA[10] and OGA/Q[11], IMCPA has a high convergence speed and avoids the prematurity to some extent. It is shown to be a novel strategy capable of solving complex machine learning tasks such as high-dimensional function numeric optimization.

1 Antibody clonal selection and immune memory

The famous antibody clonal selection theory[9] was put forward by Burnet in 1958. It establishes the idea that the antigens can selectively react to the antibodies, which are the native production and spread on the cell surface in the form of peptides. The clonal selection is a dynamic process of the immune system that self-adapts to antigen stimulation. The cells are selected when they recognize the antigens and then proliferate. When exposed to antigens, the immune cells that recognize and eliminate the antigens will be selected in the body and mount an effective response against them. The reaction leads to cell proliferating clonally and the colony has the same antibodies. Clonal selection theory thinks that some clonal cells divide into antibody producing cells, and others become immune memory cells to boost the second immune response. The information coding in memory

* Supported by National Natural Science Foundation of China (Grant Nos. 60133010 and 60372045)

** To whom correspondence should be addressed. E-mail: haifengdu72@163.com

cells makes up of immune system memory, which makes it learn, and remember the construction of protein. When exposed to the same antigens, immune cells can be selected in advance, and become activated, proliferate and implement the high quality immune function. These biological characteristics can be used in the artificial immune system.

2 Immune memory clonal programming algorithm

Without loss of generality, we consider maximizing the function (P): $\max\{f(e^{-1}(\boldsymbol{a})): \boldsymbol{a}\in\boldsymbol{I}\}$, and $\boldsymbol{x}=\{x_1, x_2, \cdots, x_m\}$ is a variable, in which, the limited-length character string $\boldsymbol{a}=a_1, a_2, \cdots, a_l$ is the antibody coding of the variable $\boldsymbol{x}$, described by $\boldsymbol{a}=e(\boldsymbol{x})$, and $\boldsymbol{x}$ is called the decoding of antibody $\boldsymbol{a}$, described as $\boldsymbol{x}=e^{-1}(\boldsymbol{a})$. The set $\boldsymbol{I}$ is called the antibody space, f is the positive real function in the set $\boldsymbol{I}$, and f is called the antibody-antigen affinity function. According to the biological terms, in the antibody $\boldsymbol{a}$, a_i is considered as the evolutionary gene called allele, whose probable values are correlated to the coding method. Generally, antibody coding is divided into m parts, and each part is l_i, where $l=\sum_{i=1}^{m} l_i$, and each part is denoted as $x_i\in[d_i, u_i]$, $i=1, 2, \cdots, m$.

In this paper, we adopt the real number encoding with a limited length. The coding length of variant x_i is determined to be l_i according to its precision. So its encoding is

$$x_i' = 0.a_{s_i+1}, a_{s_i+2}, \cdots, a_{s_i+l_i}, \tag{1}$$

where a_{s_i+j} $(j=1, 2, 3, \cdots, l_i)$ are integers from 0 to 9, s_i is the encoding origination of variant x_i, then its value is

$$x_i = (u_i - d_i)\times x_i' + d_i, \quad i = 1, 2, \cdots, n. \tag{2}$$

The antibody space is

$$\boldsymbol{I}^n=\{\boldsymbol{A}: \boldsymbol{A}=(\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_n), \boldsymbol{a}_k\in I, 1\leqslant k\leqslant n\}, \tag{3}$$

where the positive integer n is the size of antibody population, and the antibody population $\boldsymbol{A}=\{\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_n\}$ is an n-dimensional group of antibody $\boldsymbol{a}$, and it is a point in the antibody group space $\boldsymbol{I}^n$. The global optimal solutions for the set of problem P are defined as follows:

$$\boldsymbol{b}^* \equiv\{\boldsymbol{a}\in\boldsymbol{I}: f(\boldsymbol{a})=f^* \equiv\max(f(\boldsymbol{a}'): \boldsymbol{a}'\in\boldsymbol{I})\}. \tag{4}$$

For antibody population $\boldsymbol{A}$, $\vartheta(\boldsymbol{A})\equiv|\boldsymbol{A}\cap\boldsymbol{b}^*|$ denotes the number of optimal solutions in antibody population $\boldsymbol{a}$.

Essentially, the memory cell is a population too, and its antibody has the same structure as the antibody in antibody population. So, we can define memory cell as M, and the memory cell space is

$$\boldsymbol{I}^s=\{\boldsymbol{M}: \boldsymbol{M}=(\boldsymbol{m}_1, \boldsymbol{m}_2, \cdots, \boldsymbol{m}_s), \boldsymbol{m}_k\in\boldsymbol{I}, 1\leqslant k\leqslant s\}. \tag{5}$$

Then the main operation of immune memory clonal programming algorithm is:

Step 1 (Initialization): Set the halt conditions, mutation probability p_m, recombine probability p_c, clone scale n_c, clone scale of immune cell n_m, inverse probability p_i, antibody death proportion $T\%$, antibody population size n, memory cell size s, antibody coding length l, initialize the antibody population $\boldsymbol{A}(0)=\{\boldsymbol{a}_1(0), \boldsymbol{a}_2(0), \cdots, \boldsymbol{a}_n(0)\}\in\boldsymbol{I}^n$ in $[0, 1]$, select s antibodies in $\boldsymbol{A}(0)$ as memory cell $\boldsymbol{M}(0)=\{\boldsymbol{m}_1(0), \boldsymbol{m}_2(0), \cdots, \boldsymbol{m}_s(0)\}\in\boldsymbol{I}^s$, and set evolution generation $k=0$.

Step 2 (Calculate the affinity): Calculate $\boldsymbol{x}=e^{-1}(\boldsymbol{a})$ and $\boldsymbol{x}^m=e^{-1}(\boldsymbol{m})$ by Eq. (2), then calculate the affinity of antibody population and memory cell separately, marked as:

$$F^A(0):\{f(\boldsymbol{X}(0))\} = \{f(\boldsymbol{x}_1(0)), f(\boldsymbol{x}_2(0)), \cdots, f(\boldsymbol{x}_n(0))\}, \tag{6}$$

$$F^m(0):\{f(\boldsymbol{X}^m(0))\} = \{f(\boldsymbol{x}_1^m(0)), f(\boldsymbol{x}_2^m), \cdots, f(\boldsymbol{x}_s^m(0))\}. \tag{7}$$

Step 3 (Judgment of the halt conditions): If the halt conditions are satisfied, then stop, otherwise continue.

Step 4 (Refresh the antibody population T_u^C): $k=k+1$. Generate the new antibody $\boldsymbol{a}(k)$ by memory selection, namely replacing $T\%$ antibodies with a low affinity in the antibody population by the antibodies with a high affinity carried by the memory cell in the antibody population:

$$\begin{aligned}\boldsymbol{A}'(k) &= \{\boldsymbol{a}_1'(k), \boldsymbol{a}_2'(k), \cdots, \boldsymbol{a}_n'(k)\}\\ &= T_u^C(\boldsymbol{A}(k-1)\cup\boldsymbol{M}(k-1))\\ &= \{\boldsymbol{a}_1(k-1), \boldsymbol{a}_2(k-1), \cdots, \boldsymbol{a}_j(k-1),\\ &\quad \tilde{\boldsymbol{a}}_1(k-1), \tilde{\boldsymbol{a}}_2(k-1), \cdots, \tilde{\boldsymbol{a}}_l(k-1)\},\end{aligned} \tag{8}$$

where, $j+l=n$, and $l=fix(T\%\times n)$, $\tilde{\boldsymbol{A}}(k-1)\in\boldsymbol{A}(k-1)\cup\boldsymbol{M}(k-1)$, and the affinity is higher.

$fix(\cdot)$ is the function that rounds towards minus infinity. $fix(x)$ means to round the elements of x to the nearest integers towards minus infinity.

Step 5 (Inversing at $A'(k)$ T_I^C): Selecting antibodies in the antibody population as probability p_i, and select two points p and q at random. If $p < q$, then implement the inversion operating at $a_i'(k) = \{a_{i1}, a_{i2}, \cdots, a_{ip}, \cdots, a_{iq}, \cdots, a_{il}\} \in A'(k)$ as follows:

$$a_i''(k) = T_I^C(a_i'(k)) = \{a_{i1}, a_{i2}, \cdots, a_{iq}, a_{iq-1}, \cdots, a_{ip+1}, a_{ip}, \cdots, a_{il}\}. \quad (9)$$

Describe the antibody population after inversing as $A''(k) = \{a_1''(k), a_2''(k), \cdots, a_n''(k)\}$.

Step 6 (Clonal operating at $A''(k)$ T_c^C): Define

$$Y(k) = T_c^C(A''(k)) = [T_c^C(a_1''(k)), T_c^C(a_2''(k)), \cdots, T_c^C(a_n''(k))]^T, \quad (10)$$

where $T_c^C(a_i''(k)) = I_i \times a_i''(k)$, $i = 1, 2, \cdots, n$, I_i are q_i dimension row vector. $T_c^C(a_i''(k))$ is the q_i clone of antibody $a_i''(k)$.

$$q_i(k) = g(n_c, f(a_i''(k)), \Theta_i). \quad (11)$$

Θ_i indicates the affinity of antibody i to other antibodies, and we simply define it as

$$\Theta_i = \min\{D_{ij}\} = \min\{\exp(\| a_i'' - a_j'' \|)\}, \quad i \neq j;\ i, j = 1, 2, \cdots, n. \quad (12)$$

$\| \cdot \|$ is arbitrary norm, for binary coding, we use Hamming distance, but for decimal coding we generally use Euclidean distance. When we calculate Θ_i, Θ_i is generally normalized, namely $0 \leqslant \| \cdot \| \leqslant 1$, clearly, the bigger the antibody affinity is, namely the more similar, the stronger antibody-antibody restraint is, and the smaller Θ_i is. Especially when antibody affinity is 0, Θ_i is 1. Furthermore, $D = (D_{ij})_{n \times n}$, $i, j = 1, 2, \cdots, n$ is an antibody-antibody affinity matrix. D is a symmetric matrix, which denotes the diversity of population.

Generally

$$q_i(k) = \mathrm{Int}\left(n_c \times \frac{f(a_i''(k))}{\sum_{j=1}^{n} f(a_j''(k))} \times \Theta_i \right), \quad i = 1, 2, \cdots, n. \quad (13)$$

$n_c > n$ is a given value related to the clone scale, which is adjusted self-adaptively by the antibody-antigen affinity. $Int(x)$ rounds x to the least integer bigger than x. After cloned, the population becomes

$$Y(k) = \{Y_1(k), Y_2(k), \cdots, Y_n(k)\}, \quad (14)$$

where

$$Y_i(k) = \{y_{ij}(k)\} = \{y_{i1}(k), y_{i2}(k), \cdots, y_{iq_i}(k)\}$$

and

$$y_{ij}(k) = a_i''(k), \quad j = 1, 2, \cdots, q_i. \quad (15)$$

Step 7 (Immune genetic operating at $Y(k)$): Cloning of recombinant T_r^C and cloning of the mutant T_m^C are the main operators in immune genetic operating:

$$y_{ij}'(k) = T_r^C(y_{ij}(k), a_t''(k)), \quad y_{ij}(k) \in Y_i(k), \quad j = 1, 2, \cdots, q_i, \quad a_t''(k) \in A''(k), \quad i, t = 1, 2, \cdots, n \text{ and } i \neq t. \quad (16)$$

Cloning provides a full-strategy to manipulating recombination and mutagenesis, such as partial recombination and whole recombination[12], which could achieve cooperation among antibodies and increase the diversity of population, and speed up the convergence.

According to the mutation probability p_m^i, the cloned antibody populations $Y'(k)$ are mutated as $Z(k) = T_m^C(Y'(k))$.

In our algorithm, we implement mutation according to bit. Namely, for the new antibody $y_{ij}'(k)$ in the clonal population $Y'(k)$, replacing its Int(rand $*\, l$) bit code by a random integer between 0 and 9, it can be presented by

$$z_{ij}(k) = [\mathrm{Int}(\mathrm{rand} * 10) - 1] \mapsto y_{ij}'(k)^{\mathrm{Int}(\mathrm{rand} * l)}, \quad (17)$$

where $\mapsto$ means bit replacing, $y_{ij}'(k)^{\mathrm{Int}(\mathrm{rand} * l)}$ means the antibody $y_{ij}'(k)$'s Int(rand $*\, l$) bit code.

Step 8 (Clonal selection operating at $Z(k)$): $\forall i = 1, 2, \cdots, n$, if there are the mutated antibodies

$$B_i(k) = \max\{Z_i(k)\} = \{z_{ij}(k) \mid \max f(z_{ij}(k)), \ j = 1, 2, \cdots, q_i\}, \quad (18)$$

with regard to $p_s^k(Z_i(k) \cup a_i(k) \to a_i(k+1))$, we know that

$$p_s^k(\boldsymbol{a}_i(k+1)=\boldsymbol{b}_i(k))=\begin{cases}1 & f(\boldsymbol{a}_i(k))<f(\boldsymbol{b}_i(k))\\ \exp\left(-\dfrac{f(\boldsymbol{a}_i(k))-f(\boldsymbol{b}_i(k))}{\alpha}\right) & f(\boldsymbol{a}_i(k))\geqslant f(\boldsymbol{b}_i(k)) \text{ and } a_i(k) \text{ is not the best in the current antibody population}\\ 0 & f(\boldsymbol{a}_i(k))\geqslant f(\boldsymbol{b}) \text{ and } \boldsymbol{a}_i(k) \text{ is the best in the current antibody population}\end{cases} \tag{19}$$

$a>0$ is a value related with diversity of the antibody population. Generally, the better the diversity is, the bigger a is. It is obvious that the probability of keeping unchanged is $p_s^k(\boldsymbol{a}_i(k+1)=\boldsymbol{a}_i(k))=1-p_s^k(\boldsymbol{a}_i(k+1)=\boldsymbol{b}_i(k))$.

After the clonal operation, the new antibody population is $\boldsymbol{A}(k+1)$.

Step 9 (Immune operating at memory cell $\boldsymbol{M}(k)$): When compared with the antibody population, the operating on the memory cell is simple, including the following operation:

Implement the clonal operating T_c^C on memory cell $\boldsymbol{M}(k)$, clonal scale is n_m, namely

$$\begin{aligned}\boldsymbol{Y}^m(k)&=T_c^C(\boldsymbol{M}(k))\\&=[T_c^C(\boldsymbol{m}_1(k)),\ T_c^C(\boldsymbol{m}_2(k)),\ \cdots,\\&\quad T_c^C(\boldsymbol{m}_s(k))]^{\mathrm{T}}.\end{aligned} \tag{20}$$

Implement memory maturation operating T_m^C at $\boldsymbol{Y}^m(k)$, the memory cell achieves affinity maturation by increasing and optimizing the memory information. Each antibody has its own clonal space size $S_m=\dfrac{u_i-d_i}{s}$ around itself. According to the mutation probability p_m^m, the clonal antibody population $\boldsymbol{Y}^m(k)$ is mutated as $\boldsymbol{Z}^m(k)=T_m^C(\boldsymbol{Y}^m(k))$ in its clonal space. We use random mutation such as Gaussian mutation and Cauchy mutation instead of mutation by bit.

Implement clonal selection operation T_s^C at $\boldsymbol{Z}^m(k)$ and attain $\boldsymbol{B}^m(k)$, namely

$$\begin{aligned}\boldsymbol{b}_i^m(k)&=\max\{\boldsymbol{Z}_i^m(k)\}\\&=\{\boldsymbol{z}_{ij}^m(k)\mid\max f(\boldsymbol{z}_{ij}^m(k)),\ j=1,2,\cdots,q_i\},\\&\quad i=1,2,\cdots,s.\end{aligned} \tag{21}$$

Implement memory learning operating T_l^C at $\boldsymbol{B}^m(k)$: Define the windage comparison parameter δ_0 as $\dfrac{u_i-d_i}{s}$, and windage parameter is the distance of two antibodies, namely

$$\delta_j=\sqrt{\sum_{d=1}^{l}(a_{id}-a_{jd})^2}. \tag{22}$$

If optimal antibody of $\boldsymbol{A}(k+1)$ is $\boldsymbol{a}_{\mathrm{best}}(k+1)$ and the windage parameter δ_j between $\boldsymbol{a}_{\mathrm{best}}(k+1)$ and all the antibodies in $\boldsymbol{B}^m(k)$ are larger than δ_0, then we say that there is no antibody in $\boldsymbol{B}^m(k)$ that matches $\boldsymbol{a}_{\mathrm{best}}(k+1)$, put the $\boldsymbol{a}_{\mathrm{best}}(k+1)$ into the memory cell and delete the worst antibody in $\boldsymbol{B}^m(k)$. If the windage parameter δ_j between $\boldsymbol{a}_{\mathrm{best}}(k+1)$ and one antibody in $\boldsymbol{B}^m(k)$ is less than δ_0, then

$$\boldsymbol{m}_R(k+1)=\begin{cases}\boldsymbol{a}_{\mathrm{best}}(k+1) & \text{if } f(\boldsymbol{a}_{\mathrm{best}}(k+1))>f(\boldsymbol{b}_R^m(k))\\ \boldsymbol{b}_R^m(k) & \text{else}\end{cases}. \tag{23}$$

After the memory learning operation, the new memory cell is $\boldsymbol{M}(k+1)$.

Step 10 (Return to Step 3).

Additionally, there is something to be explained as the following:

(i) In this paper, the recombination strategy T_r^C can be implemented as follows:

Let $\boldsymbol{a}=\{x_1, x_2, \cdots, x_m\}$ and $\boldsymbol{b}=\{y_1, y_2, \cdots, y_m\}$ be the parent antibody. Then randomly select the recombination point $k(0<k<m)$, the clonal recombination offspring $\boldsymbol{a}'$ is computed by $\boldsymbol{a}'=\{x_1, x_2, \cdots, x_k, y_{k+1}, \cdots, y_m\}$.

(ii) Generally, the clonal mutation probability is very little, especially for the high-dimensional function optimization; the clonal mutation probability is set as the reciprocal of the function dimension.

(iii) The halt conditions are defined as the restricted iterative number or the time when the solutions are not improved at successive iterations or the two methods blending. Combining the set iterative times with hunting condition, here the algorithm is halted according to the following criterion which is

different from Ref. [12]:

$$|f^* - f^{\text{best}}| < \varepsilon, \tag{24}$$

where f^* is the global optimum of f, f^{best} is the current best function value. If $0<|f^*|<1$, the following equation holds:

$$|f^* - f^{\text{best}}| < \varepsilon |f^*|. \tag{25}$$

3 Analysis of the algorithm

3.1 Basic features of the new algorithm

When compared with evolution algorithm and the clonal selection algorithm proposed by De Castro et al.[7] immune memory clonal programming algorithm has the following features:

(i) IMCPA runs at two populations side-by-side, namely the antibody population and memory cell, which simulates the clonal selection process of immune system more comprehensively.

(ii) In IMCPA, the antibody population and memory cell evolve absolutely. For convenience, define $\cup$ for $\boldsymbol{x}=\{x_1, x_2, \cdots, x_n\}$ and $\boldsymbol{y}=\{y_1, y_2, \cdots, y_n\}$ by

$$\boldsymbol{x} \cup \boldsymbol{y} \equiv \bigcup_{i=1}^{n} \{x_i \cup y_i\}. \tag{26}$$

The evolutionary process of antibody population can be described by

$$\boldsymbol{A}(k) \cup \boldsymbol{M}(k) \xrightarrow{T_u^C} \boldsymbol{A}'(k) \xrightarrow{T_l^C} \boldsymbol{A}''(k) \xrightarrow{T_c^C} \boldsymbol{Y}(k) \xrightarrow{T_r^C \circ T_m^C} \boldsymbol{Z}(k) \cup \boldsymbol{A}(k) \xrightarrow{T_s^C} \boldsymbol{A}(k+1). \tag{27}$$

The evolution process of memory cell can be described by

$$\boldsymbol{M}(k) \xrightarrow{T_c^C} \boldsymbol{Y}^m(k) \xrightarrow{T_m^C} \boldsymbol{Z}^m(k) \cup \boldsymbol{M}(k) \xrightarrow{T_s^C} \boldsymbol{B}^m(k+1) \cup \boldsymbol{A}(k+1) \xrightarrow{T_l^C} \boldsymbol{M}(k+1). \tag{28}$$

In fact, the two processes are running side-by-side (for the sake of clarity, we describe them separately), and the main operations are similar, but the specific treatments have some difference, for example, the evolution of memory cell is implemented without immune recombination. And they have different functions: the antibody population is the basic population acting on antigens and emphasizes the global search while memory cell emphasizes the self-adaptive local search in the defining space, and achieves memory search with memory learning which keeps the diversity of population. Otherwise, memory cell helps antibodies collaborate more reasonable for each other.

(iii) Different from corresponding real number coding, IMCPA adopts a special mutation model in order to embody the antibody genetic operation better.

(iv) Clonal operation provides the chances to adopt multi-strategy at an antibody.

(v) In normal evolution algorithms, crossover is the main operator, and mutation is the background one, but clonal selection algorithm is just the opposite. The antibody population also adopts recombination operation, but the probability is 1. It is also shown in the test that the clonal selection algorithms only using mutation operator are still better than relevant evolution algorithms. Otherwise, the recombine operation can enhance the convergence speed, but it reduces the diversity of the population and enhances the prematurity.

Clonal selection algorithms and evolution algorithms are both new artificial intelligence methods inspired by evolution theory (in this paper, we think that clonal selection algorithms and evolution algorithms are to embody the microcosmic and macroscopical Darwin evolution rules separately), and both evolution and immune processes are from gene changes, so their basic operations (crossover and mutation) are similar. But through the above expression, we know that immune memory clonal programming algorithm is not the simple modification of evolutionary algorithms, and it is a new artificial immune system method.

3.2 Convergence of the new algorithm

Even though IMCPA is run at antibody population and memory cell side-by-side, but when optimal solution is achieved in the antibody population, the memory cell can also attain it by memory learning, and vice versa. So once one of the two populations is convergent, the algorithm must be convergent. According to biological terms, we think that the function of memory cell is dominating during the evolution process. So, we analyze the convergence of the new algorithm based on the memory cell.

Definition 1. $\boldsymbol{I}^{m^*}$ is called the satisfactory memory cell (antibody population) space if $\boldsymbol{I}^{m^*} = \{\boldsymbol{M} \in \boldsymbol{I}^m \mid \vartheta(\boldsymbol{M}) \geqslant 1\}$.

This definition denotes that there is at least one best antibody in the best antibody population space. And $\boldsymbol{I}^m = \boldsymbol{I}^s - \boldsymbol{I}^{m^*}$ is called the normal antibody population space. The antibody space $\boldsymbol{I}^{N_m(k)}$ resulting from the clonal operation is called clonal antibody population space, and it is a changing space, where $N_m(k) = \sum q_i(k)$.

Definition 2. For random state $\boldsymbol{M}_0$, if

$$\lim_{k\to\infty} p\{\boldsymbol{M}(k) \cap \boldsymbol{b}^* \neq \emptyset \mid \boldsymbol{M}(0) = \boldsymbol{M}_0\} = \lim_{k\to\infty} p\{\boldsymbol{M}(k) \in \boldsymbol{I}^* \mid \boldsymbol{M}(0) = \boldsymbol{M}_0\} = 1, \tag{29}$$

namely

$$\lim_{k\to\infty} p\{\vartheta(\boldsymbol{M}(k)) \geqslant 1 \mid \boldsymbol{M}(0) = \boldsymbol{M}_0\} = 1, \tag{30}$$

$\boldsymbol{M}_0$ can be convergent to $\boldsymbol{I}^{m^*}$ with probability of 1.

Theorem 1. The algorithm of IMCPA is convergent with probability of 1.

Proof. If $P_0(k) = P\{\vartheta(\boldsymbol{M}(k)) = 0\} = P\{\boldsymbol{M}(k) \cap \boldsymbol{b}^* = \emptyset\}$, according to Bayes's Theorem, we get

$$\begin{aligned} P_0(k+1) &= P\{\vartheta(\boldsymbol{M}(k+1)) = 0\} \\ &= P\{\vartheta(\boldsymbol{M}(k+1)) = 0 \mid \vartheta(\boldsymbol{M}(k)) \neq 0\} \times P\{\vartheta(\boldsymbol{M}(k)) \neq 0\} \\ &\quad + P\{\vartheta(\boldsymbol{M}(k+1)) = 0 \mid \vartheta(\boldsymbol{M}(k)) = 0\} \times P\{\vartheta(\boldsymbol{M}(k)) = 0\}. \end{aligned} \tag{31}$$

From the property of the clonal selection in clonal selection operator, the following equation holds $P\{\vartheta(\boldsymbol{M}(k+1)) = 0 \mid \vartheta(\boldsymbol{M}(k)) \neq 0\} = 0$. So

$$\begin{aligned} P_0(k+1) &= P\{\vartheta(\boldsymbol{M}(k+1)) = 0 \mid \vartheta(\boldsymbol{M}(k)) = 0\} \\ &\quad \times P_0(k). \end{aligned} \tag{32}$$

The probability of memory cell attaining the optimal solution is constituted of two parts, namely

$$\begin{aligned} &P\{\vartheta(\boldsymbol{M}(k+1)) \geqslant 1 \mid \vartheta(\boldsymbol{M}(k)) = 0\} \\ &\quad = P_{\mathrm{I}} + P_{\mathrm{II}}, \end{aligned} \tag{33}$$

where $P_{\mathrm{I}}(k)$ denotes the probability of attaining the optimal solution by memory cell and $P_{\mathrm{II}}(k)$ denotes the probability of attaining the optimal solution by antibody population. Note that $p_m^m > 0$, so $P_{\mathrm{I}}(k) > 0$. $\zeta = \min_k P(k)$, $k = 0, 1, 2\cdots$ and $P_{\mathrm{II}}(k) \geqslant 0$ then

$$P\{\vartheta(\boldsymbol{M}(k+1)) \geqslant 1 \mid \vartheta(\boldsymbol{M}(k)) = 0\} \geqslant \zeta > 0. \tag{34}$$

So

$$\begin{aligned} &P\{\vartheta(\boldsymbol{M}(k+1)) = 0 \mid \vartheta(\boldsymbol{M}(k)) = 0\} \\ &\quad = 1 - P\{\vartheta(\boldsymbol{M}(k+1)) \neq 0 \mid \vartheta(\boldsymbol{M}(k)) = 0\} \\ &\quad = 1 - P\{\vartheta(\boldsymbol{M}(k+1)) \geqslant 1 \mid \vartheta(\boldsymbol{M}(k)) = 0\} \\ &\quad \leqslant 1 - \zeta < 1. \end{aligned} \tag{35}$$

From Eq. (32) we find that

$$\begin{aligned} 0 \leqslant P_0(k+1) &\leqslant (1-\zeta) \times P_0(k) \\ &\leqslant (1-\zeta)^2 \times P_0(k-1) \cdots \\ &\leqslant (1-\zeta)^{k+1} \times P_0(0). \end{aligned} \tag{36}$$

Note that $\lim_{k\to\infty} (1-\zeta)^{k+1} = 0$, $1 \geqslant P_0(0) \geqslant 0$, so $\lim_{k\to\infty} p_0(k) = 0$. Now we can draw the conclusion that

$$0 \leqslant \lim_{k\to\infty} p_0(k) \leqslant \lim_{k\to\infty} (1-\zeta)^{k+1} P_0(0) = 0. \tag{37}$$

Then $\lim_{k\to\infty} p_0(k) = 0$. So

$$\begin{aligned} \lim_{k\to\infty} P\{\boldsymbol{M}(k) \cap \boldsymbol{B}^* \neq \Phi \mid \boldsymbol{M}(0) = \boldsymbol{M}_0\} \\ = 1 - \lim_{k\to\infty} P_0(k) = 1. \end{aligned} \tag{38}$$

The solution got by memory cell is a part of the algorithm's solution, so IMCPA can be convergent to $\boldsymbol{I}^{m^*}$ with probability 1. This completes the proof of Theorem 1.

4 Numerical experiment and results

4.1 Test functions

In order to validate our approach, the IMCPA is executed to solve the following test functions. Every one of them has lots of local optimal values, and some of them even cannot be counted indeed.

$$f_1(x) = \sum_{i=1}^{N} x_i^2, \quad -100 \leqslant x_i \leqslant 100, \quad f_{\min} = 0, \tag{39}$$

$$f_2(x) = \sum_{i=1}^{N} |x_i| + \prod_{i=1}^{N} |x_i|, \quad -10 \leqslant x_i \leqslant 10, \quad f_{\min} = 0, \tag{40}$$

$$f_3(x) = \sum_{i=1}^{N} \left(\sum_{j=1}^{i} x_j \right)^2, \quad -100 \leqslant x_i \leqslant 100, \quad f_{\min} = 0, \tag{41}$$

$$f_4(x) = \sum_{i=1}^{N} i x_i^4 + \mathrm{random}[0, 1), \quad -1.28 \leqslant x_i \leqslant 1.28, \quad f_{\min} = 0, \tag{42}$$

$$f_5(x) = \frac{1}{N} \sum_{i=1}^{N} (x_i^4 - 16x_i^2 + 5x_i), \quad -5 \leqslant x_i \leqslant 5, \quad \text{when } N = 100, \ f_{\min} = -78.33236, \tag{43}$$

$$f_6(x) = NA + \sum_{i=1}^{N}(x_i^2 - A\cos(2\pi x_i)),$$
$$-5.12 \leqslant x_i \leqslant 5.12,\ A = 10,\ f_{\min} = 0, \tag{44}$$

$$f_7(x) = -\sum_{i=1}^{N} x_i \sin(\sqrt{|x_i|}),$$
$$-500 \leqslant x_i \leqslant 500,$$
$$\text{when } N = 30,\ f_{\min} = -12569.5, \tag{45}$$

$$f_8(x) = \sum_{i=1}^{N}\frac{x_i^2}{4000} - \prod_{i=1}^{N}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1,$$
$$-600 \leqslant x_i \leqslant 600,\ f_{\min} = 0, \tag{46}$$

$$f_9(x) = -20\exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i}^{N}x_i^2}\right)$$
$$-\exp\left(\frac{1}{n}\sum_{i=1}^{N}\cos(2\pi x_i)\right) + 20 + e,$$
$$-30 \leqslant x_i \leqslant 30,\ f_{\min} = 0. \tag{47}$$

When N values are between 10 and 1000, we call them high-dimensional functions. When N is larger than 1000, we call them superhigh-dimensional functions. We solved high-dimensional functions and superhigh-dimensional functions separately by simulations.

4.2 Simulations for high-dimensional functions

In order to clarify the effect of the memory cell, in this paper we named the immune memory clonal programming algorithm without the memory cell nIMCPA.

The performance comparison of IMCPA, nIMCPA and OGA/Q[11] is shown in Table 1. The population size of IMCPA is 10 and clonal scale is 15, the memory cell size is 5 and clonal scale is 10, inverse probability p_i is 0.3, and antibody refresh proportion $T\%$ is 50%. The population size of nIMCPA is 20 and clonal scale is 30. The population size of OGA/Q is 200.

Table 1. Performance comparison of IMCPA, nIMCPA and OGA/Q

Test functions		Number of function evaluations			The optimal mean (standard)			Global minimal value
f	N	IMCPA	nIMCPA	OGA/Q	IMCPA	nIMCPA	OGA/Q	
f_1	30	2, 173	5, 117	112, 559	2.038235×10^{-7} (3.170165×10^{-7})	2.808529×10^{-7} (2.968961×10^{-7})	0 (0)	0
f_2	30	2, 729	4, 220	112, 612	2.443007×10^{-9} (3.669766×10^{-9})	3.579221×10^{-9} (3.362259×10^{-9})	0 (0)	0
f_3	30	4, 196	47, 406	112, 576	$2.575336e-010$ ($3.175653e-010$)	3.826366×10^{-10} (4.340159×10^{-10})	0 (0)	0
f_4	30	22, 402	24, 261	112, 652	4.381959×10^{-4} (2.897946×10^{-4})	4.807530×10^{-4} (3.298351×10^{-4})	6.301×10^{-3} (4.069×10^{-4})	0
f_5	100	5, 806	11, 541	245, 930	−78.33119 (2.902509×10^{-4})	−78.33009 (8.381293×10^{-4})	−78.3000296 (6.288×10^{-3})	−78.33236
f_6	30	2, 087	4, 657	224, 710	1.158469×10^{-11} (2.334151×10^{-11})	1.106457×10^{-10} (1.358627×10^{-10})	0 (0)	0
f_7	30	4, 973	9, 499	302, 166	-1.256949×10^{4} (5.946202×10^{-6})	-1.256949×10^{4} (1.252780×10^{-5})	−12569.4537 (6.447×10^{-4})	−12569.5
f_8	30	3, 516	8, 280	134, 000	1.931788×10^{-15} (2.773929×10^{-15})	5.129230×10^{-15} (3.462836×10^{-15})	0 (0)	0
f_9	30	4, 295	8, 970	112, 421	3.019807×10^{-15} (3.432247×10^{-15})	1.953993×10^{-15} (2.397899×10^{-15})	4.440×10^{-16} (3.989×10^{-17})	0

The data in Table 1 are the statistical results obtained from 50 times of random running. For f_1—f_3 and f_6, f_8, f_9 whose optimal solution is 0, the solution precisions of IMCPA and nIMCPA are slightly lower than that of OGA/Q, but the number of function evaluations reduced sufficiently. For f_4, f_5, f_7, IMCPA and nIMPCA are much better than OGA/Q, not only for the solution precisions but also for the number of function evaluations. When compared with nIMCPA, IMCPA has a higher convergence speed and solution precision.

The experimental results of IMCPA, nIMCPA and AEA[12], BGA[10] to optimize the functions f_6—f_9 are shown in Tables 2 and 3. In the tables, the data are the statistical results obtained from 10 times of random running. It can be seen that the ability of IMCPA and nIMCPA to optimize the high-dimensional functions is much better than BGA and AEA. When N is not very large, the convergence speed by IMCPA and nIMCPA is not obviously improved, or even worse than AEA (such as for Rastrigin's function f_6, when N equates 20 and 100). But when N is big enough, the improvement of convergence speed is very obvious. In addition, the performance of IMCPA is much better than nIMCPA.

Table 2. Performance comparison of IMCPA, nIMCPA, AEA and BGA for functions f_6 and f_7

N	$f_6(\varepsilon=0.1)$				f_7[a)]$(\varepsilon=10^{-2})$			
	IMCPA	nIMCPA	AEA	BGA	IMCPA	nIMCPA	AEA	BGA
20	1469	2777	1247	3608	3939	6063	1603	16100
100	4988	7221	4798	25040	11896	20978	5106	92000
200	5747	13354	10370	52948	16085	34728	8158	248000
400	12563	15475	23588	112634	26072	68556	13822	699803
1000	24408	37250	46024	337570	60720	116120	23687	/

a) For this function, the precision of IMCPA is $|f^*|$ times of BGA. "/" not tested.

Table 3. Performance comparison of IMCPA, nIMCPA, AEA and BGA for functions f_8 and f_9

N	$f_8(\varepsilon=10^{-4})$				$f_9(\varepsilon=10^{-3})$			
	IMCPA	nIMCPA	AEA	BGA	IMCPA	nIMCPA	AEA	BGA
20	2421	4657	3581	40023	1776	2768	7040	197420
100	6713	12166	17228	307625	5784	9759	22710	53860
200	8460	21577	36760	707855	9728	11305	43527	107800
400	15365	36515	61975	1600920	13915	29477	78216	220820
1000	30906	75217	97600	/	26787	33343	160940	548306

4.3 Simulation for superhigh-dimensional functions

The experimental results of IMCPA and nIMCPA to optimize the functions f_6—f_9 when N is larger than 1000 are shown in Table 4. In the table, the data are the statistical results obtained from 5 times of random running, and the simulation parameters are just the same as indicated in section 4.2. AEA cannot attain satisfactory solution in 12 hours, so we did not list it in the table. When $N>5000$, nIMCPA also cannot attain satisfactory solution in 12 hours which is denoted by "/" in the table. The simulation results show that IMCPA has also a high convergence speed for superhigh-dimensional functions which is impossible for some other algorithms.

Table 4. Performance comparison for superhigh-dimensional functions

N	$f_6(\varepsilon=0.1)$		$f_7(\varepsilon=10^{-2})$		$f_8(\varepsilon=10^{-4})$		$f_9(\varepsilon=10^{-3})$	
	IMCPA	nIMCPA	IMCPA	nIMCPA	IMCPA	nIMCPA	IMCPA	nIMCPA
2000	37879	66145	90001	157200	43003	81447	41880	54387
5000	87245	/	136869	/	125847	/	83125	/
10000	143700	/	235840	/	147037	/	138487	/

A further analysis showed that the number of function evaluations of IMCPA increases linearly with function dimension. Fig. 1 shows the analysis results for f_6, f_7, f_8 and f_9. It can be seen that when a function is one dimension higher, the average number of function evaluations increases by no more than 25.

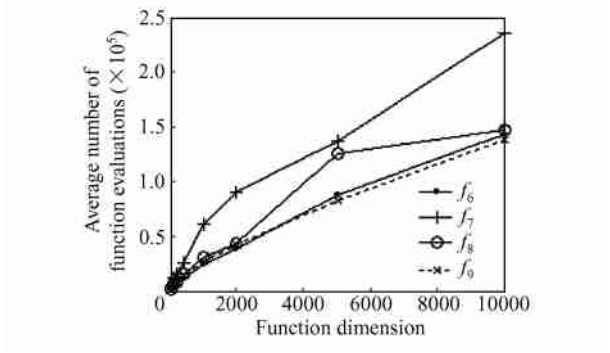

Fig. 1. The relationship between the number of function evaluations and function dimension.

In addition, our simulation results showed that, a lower death proportion, a larger population and clonal scale can enhance the diversity of antibody population further, and avoid the prematurity more effectively, but the number of function evaluations will increase too.

5 Conclusions and prospect

This paper describes in detail a new artificial immune system algorithm, namely immune memory clonal programming algorithm. Like evolutionary algorithms, this algorithm is also an achievement inspired by evolutionary theory. It is not a simple improvement of evolutionary algorithms, but a new kind of intelligent computation method. In a word, artificial immune system is not a kind of method pieced together temporarily for the reason that evolutionary algorithms have shortcomings, just like evolutionary algorithms, artificial immune system has its own bio-

logical basis, and has the completely different mechanism from evolutionary algorithms. The method has a higher convergence speed with the same computation complexity. It is shown to be a novel strategy capable of solving complex machine learning tasks such as high-dimensional function optimization.

Compared with the clonal selection algorithm proposed by De Castro et al. immune memory clonal programming algorithm simulates the memory function of immune system more systemically, but some other immune system mechanics such as bacterin, Immunodominance are not involved in this study. How to embody them in the artificial immune system algorithms is our future work.

It is necessary to note that IMCPA cannot solve very well some morbid functions such as Rosenbrock function shown below, $f(x_i)=\sum_{i=1}^{N-1}[100*(x_i^2-x_{i+1})^2+(1-x_i)^2]$, $x_i\in[-5.12, 5.12]$, $f_{\min}=0$. When $N>10$, it will get into the local optimum occasionally and is hard to break away, which will influence the convergent speed seriously. So ameliorating the algorithm inspired by immune system mechanism for these special problems is also our future work.

References

1 Dasgupta D. and Forrest S. Artificial immune systems in industrial applications. In: Proceedings of the Second International Conference on Intelligent Processing and Manufacturing of Materials. Hawaii, USA, July 10—15, 1999, 257—267.

2 Ding Y. and Ren L. Artificial immune system: theory and application. Pattern Recognition and Artificial Intelligence (in Chinese), 2000, 13(1): 52—59.

3 Cooper K. D., Hall M. W. and Kennedy K. Procedure cloning. In: Proceedings of the 1992 International Conference on Computer Languages. Oakland, California, April 20—23, 1992, 96—105.

4 Balazinska M., Merlo E. and Dagenais M. Advanced clone-analysis to support object-oriented system refactoring. In: Proceeding of Seventh Working Conference on Reverse Engineering. Brisbane, Australia, November 23—25, 2000, 98—107.

5 Esmaili N., Sammut C. and Shirazi G. M. Behavioural cloning in control of a dynamic system. In: IEEE International Conference on Systems Man and Cybernetics Intelligent Systems for the 21st Century. Vancouver, Canada, October 22—25, 1995, 2904—2909.

6 Hybinette M. and Fujimoto R. Cloning: A novel method for interactive parallel simulation. In: Proceedings of the 1997 Winter Simulation Conference. Atlanta, USA, December 7—10, 1997, 444—451.

7 De Castro L. N. and Von Zuben F. J. The clonal selection algorithm with engineering applications. In: Proceedings of Genetic and Evolutionary Computation Conference 2000, Workshop on Artificial Immune Systems and Their Applications. Las Vegas, USA, July 8—12, 2000, 36—37.

8 Kim J. and Bentley P. J. Towards an artificial immune system for network intrusion detection: an investigation of clonal selection with a negative selection operator. In: Proceedings of the 2001 Congress on Evolutionary Computation. Seoul, Korea, May 27—30, 2001, 1244—1252.

9 Lu D. and Ma B. Modern Immunology (in Chinese). Shanghai: Shanghai Scientific and Technological Education Publishing House, 1998.

10 Mühlenbein H. and Dirk S. Predictive models for the breeder genetic algorithm. Evolutionary Computation, 1993, 1(1): 25—49.

11 Leung Y. and Wang Y. An orthogonal genetic algorithm with quantization for global numerical optimization. IEEE Transactions on Evolutionary Computation, 2001, 5(1): 41—53.

12 Pan Z., Kang L. and Chen Y. Evolution Computation (in Chinese). Beijing: Tsinghua University Press, 1998.