Immune Genetic Programming based on Register-Stack Structure

Zeming Zhang, Wenjian Luo and Xufa Wang

***Abstract*—Inspired by biological immune principles, a novel Immune Genetic Programming based on Register-Stack structure (rs-IGP) is proposed in this paper. In rs-IGP, an antigen represents a problem to be solved, and an antibody represents a candidate solution. A flexible and efficient antibody representation based on register-stack structure is designed for rs-IGP. Three populations are adopted in rs-IGP, i.e. the common population, the elitist population and the self set. The immune genetic operators are also developed, including clone operator, recombination operator, mutation operator, hypermutation operator, crossover operator and negative selection operator. The experimental results demonstrate that rs-IGP has better performance.**

I. Introduction

EVOLUTIONARY algorithm is a kind of intelligent search or optimization algorithm, which simulates the biological evolutionary procedure and problems solving mechanism [1]. In this field, a broad range of methods are proposed such as genetic algorithm (GA), evolutionary strategy (ES), evolutionary programming (EP), genetic programming (GP) and gene expression programming (GEP). The fundamental differences of these algorithms are in the aspects of individual representation and particular evolutionary operators [2]. For special, GP and GEP provide creative techniques for automatically generating computer programs [2-4].

In this paper, inspired by biological immune principles, a novel immune genetic programming based on register-stack structure (rs-IGP) is proposed as an approach to generating computer programs automatically. In rs-IGP, an antigen represents a problem to be solved and an antibody in the population represents a candidate solution. The affinity between an antigen and an antibody reveals how perfect the problem is solved. A novel and efficient antibody encoding strategy based on register-stack structure is designed for rs-IGP. Three populations are involved in rs-IGP, including the common population, the elitist population and the self set. And the corresponding operators, including clone operator, recombination operator, mutation operator, hypermutation operator, crossover operator and negative selection operator, are developed in this paper. Experimental results demonstrate that rs-IGP is much more efficient than existing Immune Programming (IP) and Genetic Programming (GP).

Manuscript received February 11, 2007. This work is partly supported by the National Natural Science Foundation of China (No. 60404004), and the open foundation from Anhui Key Laboratory of Software in Computing and Communication, and the Graduate Student Innovation Foundation of USTC.

Zeming Zhang is with Nature Inspired Computation and Applications Laboratory (NICAL), Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China (e-mail: zmzhang@mail.ustc.edu.cn).

Wenjian Luo and Xufa Wang are with Nature Inspired Computation and Applications Laboratory (NICAL), Department of Computer Science and Technology, and Anhui Key Laboratory of Software in Computing and Communication, University of Science and Technology of China, Hefei 230027, China (phone: 86-551-3602824, e-mail: wjluo@ ustc.edu.cn; xfwang@ ustc.edu.cn).

II. Related Works

Currently, there are many works about Artificial Immune System (AIS). Some algorithms in AIS have been proposed, such as clone selection algorithm, negative selection algorithm, resource limited artificial immune system and so on [5-10]. They have been applied into numerous types of problems such as computer security, classification, clustering, pattern recognition and optimization [6, 9, 11-14].

However, only a few works have been concerned on automatic generation of computer programs. The typical Immune Programming (IP) algorithm is proposed by P. Musilek and his colleagues in 2005 [15]. In [15], the operators inspired by biological immune principles are introduced in detail. And a stack-based antibody representation is adopted in IP. However, the stack-based antibody representation in [15] is fixed, and this limits the algorithm much and makes the algorithm can only deal with some special kinds of simple problems.

There are also some attempts about applying some immune principles to the traditional GPs to improve their performance. In [16], for the sake of solving the problems with multimodal fitness landscapes, H. Yoshihiko extended GP by introducing immunological features so as to maintain its diversity. In [17], W. Gao introduced an improved adaptive mutation operator and an improved selection operator based on thickness adjustment mechanism in artificial immune system into the traditional evolutionary programming, and a fast immunized evolutionary programming is achieved. In [18], D. McCoy and V. Devarajan applied artificial immune systems to feature segmentation in remotely sensed images. And the results of their system are competitive with those obtained by GP. In [19], N.I. Nikolaev and his colleagues applied immune principles to traditional GP and proposed an immune version of GP, and some experiments on machine learning and time series prediction have been done to demonstrate the high performance of their improved GP .

In this paper, a new antibody representation based on register-stack structure is proposed firstly. Based on this representation, the algorithm architecture and operators are developed by using biology immune principles for reference.

III. The Immune genetic programming based on Register-Stack Structure

A. Immune Metaphor

The vertebrate immune system is composed of an innate immune system and an acquired immune system. The main role of the immune system is to discriminate "nonself" from "self" and eliminate the non-self called pathogens. Lymphocytes play an important role in acquired immune system, and the primary two kinds of lymphocytes are B-cells and T-cells [20, 21]. The immature B-cells (Pre-B) and T-cells (Pre-T) are generated in bone marrow, while mature in bone marrow and thymus respectively. During the maturation process, the B-cells or T-cells who can recognize "self" are removed. This process is called Negative Selection [7, 21]. The mature immune cells which recognize a pathogen will be activated and rapidly differentiate and proliferate [6, 20]. Therefore, the immune cells with higher and higher affinities will be obtained.

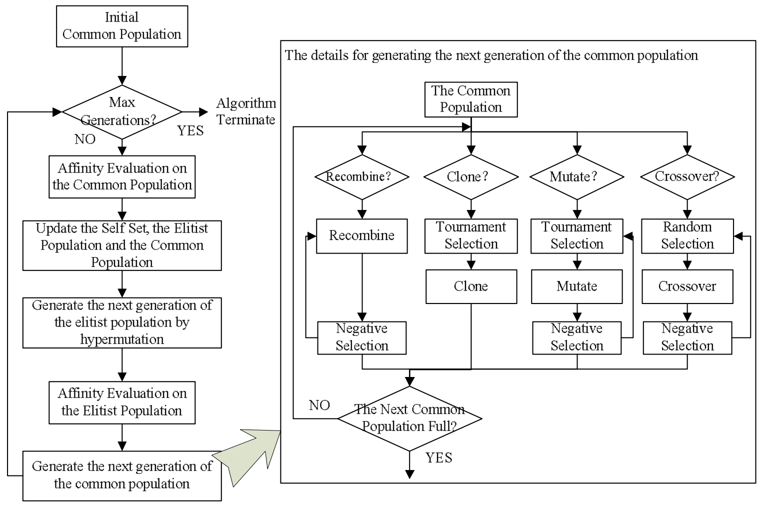
Initial Common Population
Max Generations?
NO
YES
Algorithm Terminate
Affinity Evaluation on the Common Population
Update the Self Set, the Elitist Population and the Common Population
Generate the next generation of the elitist population by hypermutation
Affinity Evaluation on the Elitist Population
Generate the next generation of the common population
The details for generating the next generation of the common population
The Common Population
Recombine?
Clone?
Mutate?
Crossover?
Recombine
Tournament Selection
Tournament Selection
Random Selection
Clone
Mutate
Crossover
Negative Selection
Negative Selection
Negative Selection
NO
The Next Common Population Full?
YES

Fig. 1. The flowchart of the immune genetic programming based on register-stack structure

B. Flowchart of rs-IGP

The flowchart of the immune genetic programming based on register-stack structure (rs-IGP) is shown in Fig.1. There are three populations in this algorithm. One is the common population, another is the elitist population and the third is the self set.

--Initially the common population consists of some randomly generated antibodies, while both the elitist population and the self set are empty.

--During the evolutionary iteration, the antibodies with highest affinities are added into the elitist population, while the antibodies with lowest affinities are added into the self set. When one antibody in common population is inserted into the elitist population or the self set, it will be removed from the common population. And a new antibody randomly generated will be added into the common population instead of the removed one.

--When the elitist population or the self set is full, the updating strategies of both the elitist population and the self set are first-in-first-out (FIFO). That is to say, the oldest individuals will be discarded.

--During the evolutionary iteration, the antibodies in the elitist population will be evolved by the hypermutation operator, while the antibodies in the common population will be evolved by operators including clone, mutation, recombination and crossover.

Therefore, the individuals in the common population are like the general immune cells in biological immune system, and the individuals in the elitist population are like the immune memory cells, while the individuals in the self set population are somewhat like the "self" in the immune system.

Different from most works about artificial immune systems with negative selection [5, 7], but similar to our previous works in [22], rs-IGP does not have a precise self definition. In fact, the worst candidate antibodies in the common population are added into the self set in every generation, while the oldest individuals in the self set are removed when the self set is full [22]. Therefore, the self set in rs-IGP is a dynamic set. However, biological immune system has a relative steady self set. To some extent, the rs-IGP can also be regarded as an improved version of traditional genetic programming algorithm. Anyway, even if the self set is not a relative steady set, the algorithm framework and some operators in rs-IGP is inspired by biological immune system. Therefore, we also regarded this rs-IGP as an immune inspired algorithm (actually called immune programming based on register-stack structure (rs-IP) in [23]).

The flowchart of the immune genetic programming based on register-stack structure is shown in Fig. 1. The detailed steps of rs-IGP algorithm are described as follows.

1) Initialization. Set the size of the common population as N. The initial common population is generated randomly. The elitist population and the self set are empty initially. Set the current evolutionary generation number $Gen = 1$.
2) If the current generation num Gen reaches the maximum generation number, go to step 8.
3) Affinity Evaluation on the common population. Calculate the affinities of the antibodies in the common population. If the optimal solution is achieved, go to step 8.
4) Update three populations. The antibodies whose affinities are no less than a threshold η_e in the common population are inserted into the elitist population, and the antibodies whose affinities are no more than a threshold η_s are inserted into the self set. And these antibodies in the common population are replaced by new antibodies randomly generated. The new antibodies should suffer the negative selection process, i.e., the new antibodies should not match any individual in the self set. In addition, the updating strategies of both the elitist population and the self set are first-in-first-out (FIFO).
5) Generate the next generation of the elitist population by hypermutation. Every antibody in the elitist population will generate several antibodies with hypermutation until a better one is generated. And this better antibody will replace the original one. In order to save computation time, the hypermutation is retried no more than predefined times and then ignored, leaving the original antibody in the elitist population. If the optimal solution is achieved, go to step 8.
6) Generate the next common population.

6.1) One operator is selected from four operators (recombination, clone, mutation and crossover), and used to generate one antibody in the next common generation.

6.1.1) Recombination. A new antibody is randomly generated and added into the next common population.

6.1.2) Clone. An antibody is selected from the common population according to the tournament selection, and added into the next common population.

6.1.3) Mutation. An antibody is selected from the common population according to the tournament selection. And this antibody mutates to a new antibody. This new antibody is added into the next common population.

6.1.4) Crossover. One antibody is randomly selected from the common population; the other is randomly selected from the elitist population. With the crossover operation on these selected two antibodies, two new antibodies are generated. Randomly select one from these two new antibodies, and add it into the next common population.

6.1.5) All new generated antibodies by recombination, mutation and crossover should suffer the negative selection process. If one antibody can not pass the negative selection process, it should be regenerated.

6.2) If the size of the next common population does not reach N, go to step 6.1.

7) $Gen = Gen + 1$, go to step 2.
8) End.

From the flowchart in Fig. 1, there are two ways to obtain the optimal solution. One is through the hypermutation operation on the elitist population, and the other is through mutation, crossover or recombination operations on the common population.

In rs-IGP, the elitist population consists of some elitist antibodies with good affinities. The hypermutation operation on the antibodies in elitist population can strengthen neighborhood searching ability of rs-IGP, and can fasten its convergence speed.

The common population consists of some antibodies with general affinities. The recombination, mutation, clone and crossover operations on the antibodies in common population can maintain the population diversity and enlarge the searching space.

The self set consists of some antibodies with bad affinities. This is used in the negative selection process. In this process, the antibody who tightly matches any one in the self set will be removed. The negative selection process can avoid generating too many useless antibodies and force the algorithm towards the good direction. The partial matching rules between the self individual and the antibody could be Hamming distance or r-continuous-bits [24]. In this paper, the Hamming distance is adopted as the partial matching rule.

C. Genotype and Phenotype of the Antibody

The antibody representation is the foundation of the immune genetic operators. In [25], J. Devaney proposed the representation principle "*Once a problem is described using an appropriate representation, the problem is almost solved.*" So the representation is very important for solving a problem and has much effect on the algorithm performance.

The most famous algorithms for computer program's automatically generation are genetic programming (GP) and gene expression programming (GEP). In GP, the genotype is mainly based on tree or S-expression [4, 25]. While in GEP, the genotype is based on K-expression [2, 3]. For this kind of algorithms, the phenotype is a piece of computer program. For simple, it can be an arithmetic expression.

There are several attempts to apply the stack-based structure to represent the individual (i.e. the candidate solution) [15, 26]. Stack-based machines have a small size, low system complexity and high system performance [27]. Because all the operands are located and accessed on the stack, the instructions are short and simple [15, 27]. In [15], the immune programming (IP) are demonstrated using stack-based examples, where both the stack depth and the stack data are fixed. This makes the IP too difficult to solve complex problems. One significant drawback of stack-based machines is that it's very difficult to visit the data deep in stack, unless the instruction "duplicate the *N*th data element" provided [27].

TABLE I. ONE EXAMPLE OF THE INSTRUCTION SET [15, 27]

Instruction	Code	Description	Stack	Register
Stack Operate Instruction				
DUP	A	Duplicate the top of the stack	x x y	z
SWAP	B	Swap the top two elements of the stack	y x	z
OVER	C	Duplicate the 2nd element of the stack	y x y	z
NOP	D	No operation	x y	z
Stack Arithmetic Instruction				
S-ADD	E	Add the top two elements of the stack	x+y	z
S-MUL	F	Multiply the top two elements of the stack	x*y	z
Register-Stack Arithmetic Instruction				
RS-ADD	G	Add the top of the stack and the register	y	x+z
RS-MUL	H	Multiply the top of the stack and the register	y	x*z

For flexibility, in rs-IGP the genotype is based on register-stack structure. The instruction set adopted in our rs-IGP experiments is given in Table Ⅰ. Besides the stack instructions mentioned in [15, 27], some register-stack instructions are appended. The operand set consists of the variants of the problem to be solved. For convenience, each instruction is represented by a single upper letter. In Table Ⅰ, to illustrate the functionalities of the instructions, it is supposed that the initial data in stack and register is In and {z} respectively. And the data changes brought by the instructions are shown in the "Stack" and "Register" columns in Table Ⅰ.

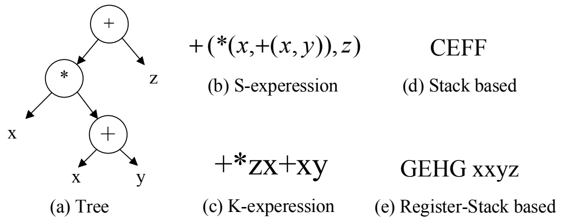

Fig. 2. Different representations for the phenotype x^2+xy+z : (a) Tree representation; (b) S-expression; (c) K-expression; (d) Stack based representation; (e) Register-Stack based representation

For the same phenotype, arithmetic expression x^2+xy+z, several different representations are shown in Fig. 2. Notably, as for the stack-based representation adopted in [15], the stack elements are fixed, i.e.and the default operand of each instruction is the top of the stack. So, the individual representation does not include the operands.

The antibody of rs-IGP consists of an instruction head and an operand tail. The instruction head consists of symbols coming from the instruction set. The operand tail consists of symbols only coming from the operand set, and is stored in the stack. Because only one register is used and the register data is not initialized, the register data does not encoded into the antibody representation. For these instructions introduced in Table Ⅰ, one instruction most consumes one operand. So, when $T \geq H+1$, every instruction in the head of antibody can be executed successfully. In which, T denotes the tail length and H denotes the head length. In general, there will be some redundant operands in the tail. The instructions in the head of an antibody are executed one by one, and the result is obtained in the register.

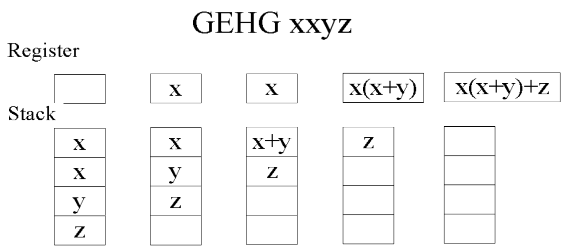

Fig. 3. An example of translation from genotype (GEHG xxyz) into phenotype

One example is shown in Fig.3. Especially, when the first register-stack arithmetic instruction executes, the top element of the stack will be directly moved to the register because the register data is not initialized.

D. *Affinity Evaluation*

In rs-IGP, the antigen represents the problem to be solved. The problem can be described in different ways. One is to provide the pairs of inputs and outputs. The other is to provide the arithmetic expression directly. However, it is difficult to evaluate the similarity of two arithmetic expressions directly because one arithmetic expression has many different forms. Therefore, to compute the affinity, the pairs of inputs and outputs values are sampled according to the given arithmetic expression. That is to say, the inputs of the arithmetic expression are sampled and the corresponding outputs are calculated. These pairs represent the relation requested by the arithmetic expression to be solved.

For example, the antigen is x^2+xy+z, and five numerical values are sampled for each input {(x, y, z)} = {(48, 0, 61), (25, 52, 2), (88, 29, 3), (26, 95, 19), (34, 4, 9)}. The corresponding output values of the antigen are {2365, 1927, 10299, 3165, 1301}. Because the number of the sampling values of inputs is limited, maybe the pairs can not completely represent the relation requested. But when the sampling space is large and the data distribution is appropriate, the relation requested by the arithmetic expression can be completely represented.

For an antibody, the operands are replaced by the corresponding sampling values and the output is calculated by executing the instructions in the antibody one by one, as described in part C of section Ⅲ. As for the antibody GEHG yyxzx, if five input samples are {(x, y, z)} = {(48, 0, 61), (25, 52, 2), (88, 29, 3), (26, 95, 19), (34, 4, 9)}, the corresponding outputs are {61, 4006, 3396, 11514, 161}.

The affinity between an antigen and an antibody reveals how perfect the problem is solved. As for rs-IGP, the affinity is calculated according to the difference between the outputs of one antibody and the expected outputs. For convenience, it should be normalized. The affinity evaluation function adopted in this paper is shown as equation (1).

$$Affinity\ (A)=\begin{cases}1-\dfrac{\sum_{i=1}^{n}\dfrac{|a_i-e_i|}{|e_i|}}{n} & (\forall i, |a_i-e_i|\le|e_i|)\\ 0.05 & (\exists i, |a_i-e_i|>|e_i|)\end{cases} \quad (1)$$

In this equation, n denotes the number of the sampling input/output pairs, e_i represents the *i*th expected output, and a_i represents the ith output of the antibody *A*. According to equation (1), higher the affinity is, more perfect the problem is solved. And when the output of the antibody is far away from the expected output, the affinity value is set as a small value, i.e. 0.05.

E. Operators

In rs-IGP, there are six primary operators: hypermutation, recombination, clone, mutation, crossover and negative selection. Among these operators, the hypermutation is conducted on the elitist population, while other operators are conducted on the common population.

Hypermutation: At every generation, each antibody in the elitist population will be hypermutated to generate a better one. Each gene in the antibody has a probability with P_{ghm} to be mutated. The hypermutation operator is retried again and again until a better antibody is achieved. For each antibody, the maximum number of new antibodies is denoted by the parameter *HyperNum* . If no better antibody is achieved after *HyperNum* antibodies are generated, keep the original one. The hypermutation operator can strengthen the neighborhood searching ability and can fasten the convergence speed.

One operator is selected from recombination, clone, mutation and crossover, and used to generate one antibody in the next common generation. The selection probability of recombination operator, clone operator, mutation operator and crossover operator is denoted as P_r , P_c , P_m and P_o respectively ($P_r+P_c+P_m+P_o=1$).

Recombination: A new antibody will be generated through recombination operator by randomly selected genes. In order to keep the antibody's validity, the genes in the head of the antibody can only be selected from the instruction set, and the genes in the tail only from the operand set. This recombination operator can improve the population diversity.

Clone: The clone operator just copies the selected antibody to the next common population directly. The selection strategy adopted in this paper is tournament selection. Other selection strategy such as roulette wheel selection can be adopted too. This clone operator can keep the good individuals in the population.

Mutation: As for the mutation operator, every gene in the antibody has a certain probability P_{gm} to be mutated. One gene can only mutate to a gene with the same type. This means that a gene in the head of the antibody can only mutate to a gene in the instruction set, and a gene in the tail can only mutate to a gene in the operand set. Similar to clone operator, the tournament selection strategy is adopted in this paper to select the antibody. Therefore, this mutation operator gives chance to the antibodies with good affinities to be better antibodies.

Crossover: As for crossover operator, two parents are selected randomly, one from the elitist population and the other from the common population. Then the one-point crossover strategy is adopted. One of the children will be selected randomly and added into the next common population. This crossover operator makes the children keep some better pieces of the elitist antibody.

Negative selection: The negative selection operator is adopted to filter the new antibodies. If the Hamming distance between a new antibody and any self individual is less than the matching threshold, this new antibody will be removed. That is to say, if the number of identical genes in the corresponding positions is no less than the matching threshold, this new antibody will be removed. This negative selection operator can delete useless antibodies and can force the algorithm towards the good direction.

For convenience, the matching degree of any two individuals is calculated as $\frac{l_{AB}}{L}$, where l_{AB} denotes the number of identical genes in the corresponding positions of two individuals *A* and *B*, and *L* denotes the length of the individual. And the matching threshold η_m is set as 0.7 in this paper. As for a new antibody A and a self individual B, if $\frac{l_{AB}}{L}$ is no less than η_m, these two individuals match. And the new antibody A will be removed.

IV. Experiments and Discussions

In this section, the proposed rs-IGP is tested by some arithmetic expressions. All arithmetic expressions used in [15] are considered in this paper. Besides this, the comparative experiments of rs-IGP, IP and GP are conducted, and some more complex experiments are conducted too.

In this paper, the fixed parameters in all the experiments are listed as follows: $HyperNum = 10$, $P_{ghm} = 0.03$, $P_{gm} = 0.06$, $P_r = 0.3$, $P_c = 0.1$, $P_m = 0.4$, $P_o = 0.2$, $\eta_e = 0.7$, $\eta_s = 0.3$, $\eta_m = 0.7$. In addition, as adopted in [15], the number of the input/output samples is 5.

The head length of the antibody is H, the tail length is $H+1$. So the antibody length is $2H+1$. The size of common population is N, the size of the self set is the same as the common population, and the size of the elitist population is half of the common population.

Each experiment is run 10 times independently. All experiments are run on a personal computer with Intel 3.0GHz CPU and 1G DDR RAM.

A. Simple Experiments

In this subsection, different kinds of arithmetic expressions are considered. In this section, the size of the common population is set to 20. The experimental results for these arithmetic expressions are shown in Table Ⅱ.

In Table Ⅱ, the "Generation" means the average generation of 10 times experiments, and the number in brackets is the standard deviation. The "Time" is the total executing time for 10 times experiments. The unit of time is second and its precision in our experiments is one second.

From Table Ⅱ, it can be observed that all these simple problems can be solved perfectly almost in one second. Therefore, the rs-IGP algorithm proposed in this paper is a very efficient algorithm. It is noted that these simple expressions are also tested in [15], and the reulsts are almost equivalent to those given in Table Ⅱ. However, these arithmetic expressions are too simple. In part B of section Ⅳ, a more complex arithmetic express in [15] is adopted in the comparative experiments.

TABLE Ⅱ. EXPERIMENTAL RESULTS FOR FIVE SIMPLE ARITHMETIC EXPRESSIONS

x^8	Head Length	9	10	11	12	13
	Generation	76.9(46.5)	24.3(22.1)	16(17.9)	14.5(9.5)	9.9(11)
	Time	12	6	3	3	3
$xy + y^2 + z$	Head Length	6	7	8	9	10
	Generation	24.4(19.9)	16.8(10.5)	27(23.7)	18.5(15.8)	13.9(11.6)
	Time	3	2	4	4	3
$(xy)^2$	Head Length	4	6	8	10	12
	Generation	13.3(15.9)	2.9(2.1)	4(3.8)	4.7(5.2)	5.8(5.7)
	Time	1	1	1	1	2
$(x+y)^2$	Head Length	4	6	8	10	12
	Generation	17.9(16.1)	8.9(6.7)	6.1(6.8)	7.1(5.2)	11(8.7)
	Time	1	1	1	2	2
$x^2 + y^2 + x + y$	Head Length	8	10	12	14	16
	Generation	25.7(14.9)	15.7(12.9)	11.6(7.8)	17.2(15.7)	12.1(6.6)
	Time	4	3	3	4	4

B. Comparison of rs-IGP, IP and GP

In this subsection, the performance comparisons of rs-IGP, IP in [15] and GP have been done. In [15], the stack data are fixed and the antibody consists of only symbols coming from the instruction set. This is similar to the instruction head of the antibody in rs-IGP. So, the head length of the antibody in rs-IGP is set the same size as the antibody length in IP [15]. In addition, GP used to be compared in this section is a stack-based version mentioned in [15, 26].

The arithmetic expression considered in this experiment is $(x+y)^3$. The antibody's length in IP and GP is 10, and the antibody's head length in rs-IGP is 10 too. The stopping criteria is either an optimum solution is found or the maximum generation is reached. The maximum generation $G_{\max}$ is set to 1000. Each experiment has been done 10 times independently. The average generation number and the success ratio are shown in Table Ⅲ. The number in brackets is the standard deviation. In addition, as for rs-IGP, the "**Population Size**" in Table Ⅲ means the size of the common population. Because the affinities of the individuals in both the common population and the elitist population of rs-IGP are needed to be evaluated, and the fitness evaluation is the most complex operator, the number of fitness evaluations is also given in Table III for fair comparisons, denoted by **NAE**.

In Table Ⅲ, the results of GP and IP are taken from [15]. There are no standard deviations for the average generations and the number of affinity evaluations in [15]. As for IP and GP in [15], the number of affinity evaluations in Table Ⅲ is estimated by the product of "***Average Generation* * *Population Size***".

From Table Ⅲ, it is demonstrated that the rs-IGP is clearly superior to IP and GP in average generations, average number of affinity evaluations and success ratio. Even if the common population size is small, i.e., 10, the success ratio of rs-IGP still can reach 100%. There are three primary reasons: (1) An elitist population is maintained and a hypermutation operator on the elitist population is adopted in rs-IGP. This can strengthen the neighborhood searching ability much and can fasten the convergence speed. (2) The representation of antibody in rs-IGP has better flexibility. (3) The common population in rs-IGP has better diversity. This is maintained by the recombination operator, clone operator, mutate operator and crossover operator.

TABLE Ⅲ. COMPARISON OF RS-IGP, IP AND GP

Population Size		10	50	100	200	300	400	500
GP [15]	Generation	N/A	468	266	103	130	130	32
	NAE*	N/A	23400	26600	20600	39000	52000	16000
	Success Ratio	0%	40%	60%	60%	90%	90%	90%
IP [15]	Generation	553	76	30	28	17	18	19
	NAE	5530	3800	3000	5600	5100	7200	9500
	Success Ratio	90%	100%	100%	100%	100%	100%	100%
rs-IGP	Generation	52.7(24.4)	9.9(12.1)	7.3(7.7)	2.8(1.8)	4(3.2)	3.5(3.2)	2.4(1.7)
	NAE	2337(1570)	2304(3315)	2070(2149)	1625(924)	3654(3186)	4444(3992)	3408(1841)
	Success Ratio	100%	100%	100%	100%	100%	100%	100%

*NAE means the number of affinity evaluations.

C. More Complex Experiments

The immune programming algorithm proposed in [15] is very limited for its fixed stack structure. So, IP can only solve some special kinds of simple problems. For example, the simple arithmetic expression $ab+cd$ can not be solved through IP. But this problem can be easily solved through rs-IGP proposed in this paper, because the representation of antibody in rs-IGP has more flexibility. And rs-IGP can solve more complex problems, such as $a^2+b^2+c^2+ab+bc+ca$. These two examples are adopted in this section and the experimental results are shown in Table Ⅳ.

Each experiment has been run 10 times independently, and the "Generation" in Table Ⅳ means the average value of the 10 times experimental results. The number in brackets is the standard deviation. Noted that if one experiment does not achieve the optimal solution before the maximum generation, the iteration number is not counted into the final average value (i.e., "Generation" in Table Ⅳ).

TABLE Ⅳ. EXPERIMENTAL RESULTS FOR EXPRESSIONS $ab+cd$ AND $a^2+b^2+c^2+ab+bc+ca$

$ab+cd$					
Head Length	6	8	10	12	14
Generation	50.8(36.4)	79.6(87.9)	46.9(29.9)	113.3(89.3)	74(57.8)
Success Ratio	100%	100%	100%	100%	100%
$a^2+b^2+c^2+ab+bc+ca$					
Head Length	10	12	14	16	18
Generation	3212.8(3859.8)	6055.8(3244.4)	3956.4(3164.7)	4272.4(3229.7)	2952(2832.8)
Success Ratio	50%	60%	90%	100%	100%

For the expression $ab+cd$, the maximum generation $G_{\max}$ is set to 300. While for $a^2+b^2+c^2+ab+bc+ca$, G_{mzx} is set to 10000. The common population size *N* is set to 20 in each experiment. One solution for expression $ab+cd$ with head length 5 is "BHHFGdcabdb", and another with head length 10 is "CDEGEHFGFGbacccbbcccc".

The experimental results shown in Table Ⅳ demonstrate that rs-IGP can solve some complex problems. When the antibody length increases, the number of satisfied solutions increases. So, this makes it easier to find a satisfied solution. But at the same time, the searching space also increases. This will increase the difficulty of searching for a satisfied solution. This is a dilemma. Therefore, a better algorithm performance will be achieved if the antibody length is appropriately set. Furthermore, other parameters such as P_r, P_c, P_m, η_e, η_s also can be tuned to achieve a better performance. These will be our further works.

V. CONCLUSIONS

In this paper, a novel Immune genetic programming based on Register-Stack Structure, namely rs-IGP, is proposed. The rs-IGP is inspired by biology immune principles and can automatically generate computer programs, no need domain knowledge and no need human's interaction. The antibody representation based on register-stack structure is given in this paper. The operators including hypermutation, recombination, clone, mutation, crossover and negative selection are designed. The experimental results show that rs-IGP is a very efficient approach to automatically generating computer programs, and its performance is much better than those of both IP and GP.

REFERENCES

[1] T. M. Mitchell, *Machine Learning*. NewYork, USA: McGraw-Hill, 1997.

[2] C. Ferreira, "Gene Expression Programming: A New Adaptive Algorithm for Solving Problems," *Complex Systems,* vol. 13, pp. 87-129, 2001.

[3] C. Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*. NewYork, USA: Springer-Verlag, 2006.

[4] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[5] L. N. Castro and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*. London: Springer-Verlag, 2002.

[6] L. N. d. Castro and F. J. V. Zuben, "Learning and Optimization Using the Clonal Selection Principle," *IEEE Transactions on Evolutionary Computation,* vol. 6, pp. 239-251, 2002.

[7] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri, "Self-Nonself Discrimination in a Computer," in *1994 IEEE Symposium on Research in Security and Privacy*, Los Alamitos, CA, 1994, pp. 202-212.

[8] J. Kim and P. J. Bentley, "Immune Memory and Gene Library Evolution in the Dynamical Clonal Selection Algorithm," *Journal of Genetic Programming and Evolvable Machines,* vol. 5, pp. 361-391, 2004.

[9] D. Dasgupta, J. Zhou, and F. Gonzalez, "Artificial Immune System (AIS) Research in the Last Five Years," in *the 2003 Congress on Evolutionary Computation (CEC '03)*, Canberra, Australia, 2003, pp. 123-130.

[10] J. Timmis and M. Neal, "A Resource Limited Artificial Immune System for Data Analysis," *Knowledge Based Systems,* vol. 14, pp. 121-120, June 2001.

[11] E. Hart and J. Timmis, "Application Areas of AIS: The Past, the Present and the Future.," in *the 4rd International Conference on Artificial Immune Systems (ICARIS-2005)*, 2005, pp. 483-497.

[12] J. Kim, W. Wilson, U. Aickelin, and J. McLeod, "Cooperative Automated Worm Response and Detection Immune Algorithm(CARDINAL) Inspired by T-cell Immunity and Tolerance," in *4th International Conference on Artificial Immune Systems (ICARIS 2005)*, Banff, Alberta, Canada, 2005, pp. 168-181.

[13] J. Zhou and D. Dasgupata, "Augmented Negative Selection Algorithm with Variable-Coverage Detectors," in *the 2004 Congress on Evolutionary Computation (CEC '04)*, 2004, pp. 1081-1088.

[14] U. Aickelin, J. Greensmith, and J. Twycross, "Immune System Approaches to Intrusion Detection - A Review," in *3rd International Conference on Artificial Immune Systems (ICARIS 2004), LNCS 3239*, Catania, Italy, 2004, pp. 316-329.

[15] P. Musilek, A. Lau, M. Reformat, and L. Wyard-Scott, "Immune Programming," *Information Sciences,* vol. 176, pp. 972-1002, 2005.

[16] Y. Hasegawa and H. Iba, "Multimodal search with immune based genetic programming," in *the 3rd International Conference on Artificial Immune Systems (ICARIS 2004)*, Catania, Italy, 2004, pp. 330-341.

[17] W. Gao, "Fast immunized evolutionary programming," in *Congress on Evolutionary Computation*, San Diego, CA, USA, 2004, pp. 666-670.

[18] D. F. McCoy and V. Devarajan, "Artificial immune systems and aerial image segmentation," in *IEEE International Conference on Systems, Man, and Cybernetics*, Florida, USA, 1997, pp. 867-872

[19] N. I. Nikolaev, H. Iba, and V. Slavov, "Inductive genetic programming with immune network dynamics," in *Advances in genetic programming*. vol. 3 Cambridge, MA, USA: MIT Press, 1999.

[20] F. M. Bumet, *The Clonal Selection Theory of Acquired Immunity*. Nashville, Tennessee: Cambridge University Press, 1959.

[21] L. N. d. Castro and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*. London: Springer-Verlag, 2002.

[22] Y. Zhang, W. Luo, Z. Zhang, B. Li, and X. Wang, "A Hardware-Software Partitioning Algorithm Based on Artificial Immune Principles," *Applied Soft Computing,* accepted in March, 2007.

[23] Z. Zhang, "Artificial Immune Algorithms and Their Applications," in *Ph.D Thesis*. vol. Ph.D Hefei, China: University of Science and Technology of China, 2007.

[24] W. Luo, Z. Zhang, and X. Wang, "A Heuristic Detector Generation Algorithm for Negative Selection Algorithm with Hamming Distance Partial Matching Rule," in *5th International Conference on Artificial Immune Systems (ICARIS 2006), LNCS 4163*, Instituto Gulbenkian de Ciência, Oeiras, Portugal, 2006, pp. 229-243.

[25] J. Devaney, J. Hagedorn, O. Nicolas, G. Garg, A. Samson, and M. Michel, "A Genetic Programming Ecosystem," in *Proceedings of the 15th International Parallel and Distributed Processing Symposium* Los Alamitos, CA, USA, 2001, pp. 1323-1330.

[26] T. Perkis, "Stack-based genetic programming," in *the IEEE World Congress on Computational Intelligence*, Piscataway, NJ, USA, 1994, pp. 148-153.

[27] P. J. Koopman, *Stack computers: the new wave*. New York, USA Halsted Press, 1989.