Controlled Gene-Accumulation Programming

Najla Akram AL-Saati **Nidhal Al-Assady**

dr.najla_alsaati@uomosul.edu.iq

College of Computer Sciences and Mathematics / University of Mosul

Received on: 25/09/2008 **Accepted on: 04/12/2008**

ABSTRACT

A full description of a new novel method proposed for Automatic Programming is brought forward in this work. Controlled Gene-Accumulation programming is a method that is purely inspired by concepts of nature. Research will show that this method provide a better overall performance especially due to the isolation of terminals from functions. Chromosome flip folding is a new crossover operator introduced in this work; it will prove to be efficient in introducing new genetic material.

A new stage is added to the evolutionary process along with mutation, transposition and recombination, the stage is inspired from natural inoculation with two new operators: ***vaccines*** and ***serums***, these two operators proved to have a huge effect on evolving systems by ensuring the death of weak individuals and survival of the fittest with the addition of enforced immunity of the system.

Investigations of this approach proved its superiority over other methods in various aspects, it is in a way controlled to adapt to rule complexity and the production of minimal chromosomal encodings as the chromosomes all have varying lengths genes. It has a much faster execution time compared to the well-known Gene Expression Programming method.

Keywords*: Automatic Programing, Controlled Gene-Accumulation Programming, Gene expression Programing.*

البرمجة بتراكم الجينات المقاد تحت السيطرة

نجلاء اكرم الساعاتي **نضال حسين الاسدي**

كلية علوم الحاسوب والرياضيات / جامعة الموصل

تاريخ استلام البحث: 2008/09/25 **تاريخ قبول البحث: 2008/12/04**

الملخص

يقدم هذا البحث وصف كامل لنموذج مبتكر جديد مقترح للبرمجة الالية. البرمجة بتراكم الجينات المسيطر عليه هي طريقة مستوحاة من حقائق الطبيعة بشكل كامل وسيبين البحث ان هذه الطريقة توفر اداء كلي افضل وخاصة بسبب عزل الطرفيات (terminals) عن الدوال. تم ايضا تقديم معامل جديد يدعى (Chromosome flip-folding) والذي اثبت كفاءة بتقديم مادة جينية جديدة.

تم ايضا تقديم مرحلة جديدة للعملية التطورية بالاضافة الى الطفرات الوراثية ، العمليات التنظيمية وعمليات المزج. استوحيت فكرة هذه المرحلة من فكرة اللقاح الطبيعي وتتضمن معاملين: اللقاح والمصل، حيث اثبت المعاملان اثرهما الكبير على النظم الخاضعة للتطور وذلك من خلال ضمان موت الافراد الضعيفة وبقاء الافضل هذا بالاضافة الى توفير المناعة المعززة للنظام.

ادت الاستقصاءات التي تمت على هذا النموذج الى اثبات تفوقه على كل الطرق الاخرى من عدة نواحي، وهو مقاد بشكل يتلائم مع تعقيد القوانين ويؤدي الى انتاج كروموسومات ذات اقل قدر ممكن من الترميز وذلك بسبب اختلاف اطوال الجينات في كل كروموسوم. للنموذج سرع تنفيذ عالية مقارنة بالطريقة المعروفة (Gene Expression Programming).

الكلمات المفتاحية: *البرمجة الذاتية ، البرمجة بتراكم الجينات المقاد تحت السيطرة ، البرمجة بالتمثيل الجينـي.*

1. Introduction

Controlled Gene-Accumulation Programming (CGAP)[1], is a new genotype/ phenotype system that uses the idea of evolution to generate computer programs to solve real-world problems. This method is inspired by natural chromosomes and is the closest to represent them among most of the proposed Linear GP variants [2, 4, 7, 8]. This method employs a multi-varying gene system where rules are represented by expression trees encoded as double-stranded linear entities in a close similitude to natural DNA strands. These multi-varying gene chromosomes have the ability to represent any expression tree whatever its complexity.

CGAP is a genetic system, that employs populations of individuals, it applies selection schemes to choose individuals according to their fitness measures, introduces genetic variations by one or more of the available genetic operators to achieve genetic diversity in evolving populations, and it utilizes a fitness measure to evaluate these individuals.

The elementary difference among GAs, GP and CGAP algorithms exist in the encoding of individuals: GAs encode the individuals in linear strings of fixed length called chromosomes; in GP they are non-linear entities of different sizes and shapes called parse trees; and in CGAP they are represented as fixed-length linear strings called genomes or chromosomes and are then expressed as non-linear entities of different sizes and shapes.

2. CGAP Algorithm

Evolution in CGAP is achieved through a number of successive generations, it begins with the creation of individuals in the first population by filling up the two-stranded genes of each chromosome in the population with randomly generated functions and terminals respectively taken from the predefined sets of functions and terminals. Then each chromosome is expressed into its corresponding phenotype structure. Afterwards, the selection process is carried out based on Roulette wheel selection scheme coupled with elitism, where individuals are selected to go through mutation, transposition and recombination according to their fitness to form a new generation which, in turns, undergoes the same process. The succession is repeated until the termination criterion is met. Figure (1) gives the CGAP flowchart.

The newly added stage of inoculation is applied only twice according to predefined probabilities. The vaccine inoculation is applied in the early stages of evolution (usually the second generation), while the serum inoculation is applied at the last stages, unlike other operators, which are probabilistically applied in all generations. After reproduction no editing, by any means, of the resulting individual is necessary before fitness evaluation, as all the resulting members of the population are correctly synthesized programs in all cases, this is ensured by the nature functionality of the genetic operators used.

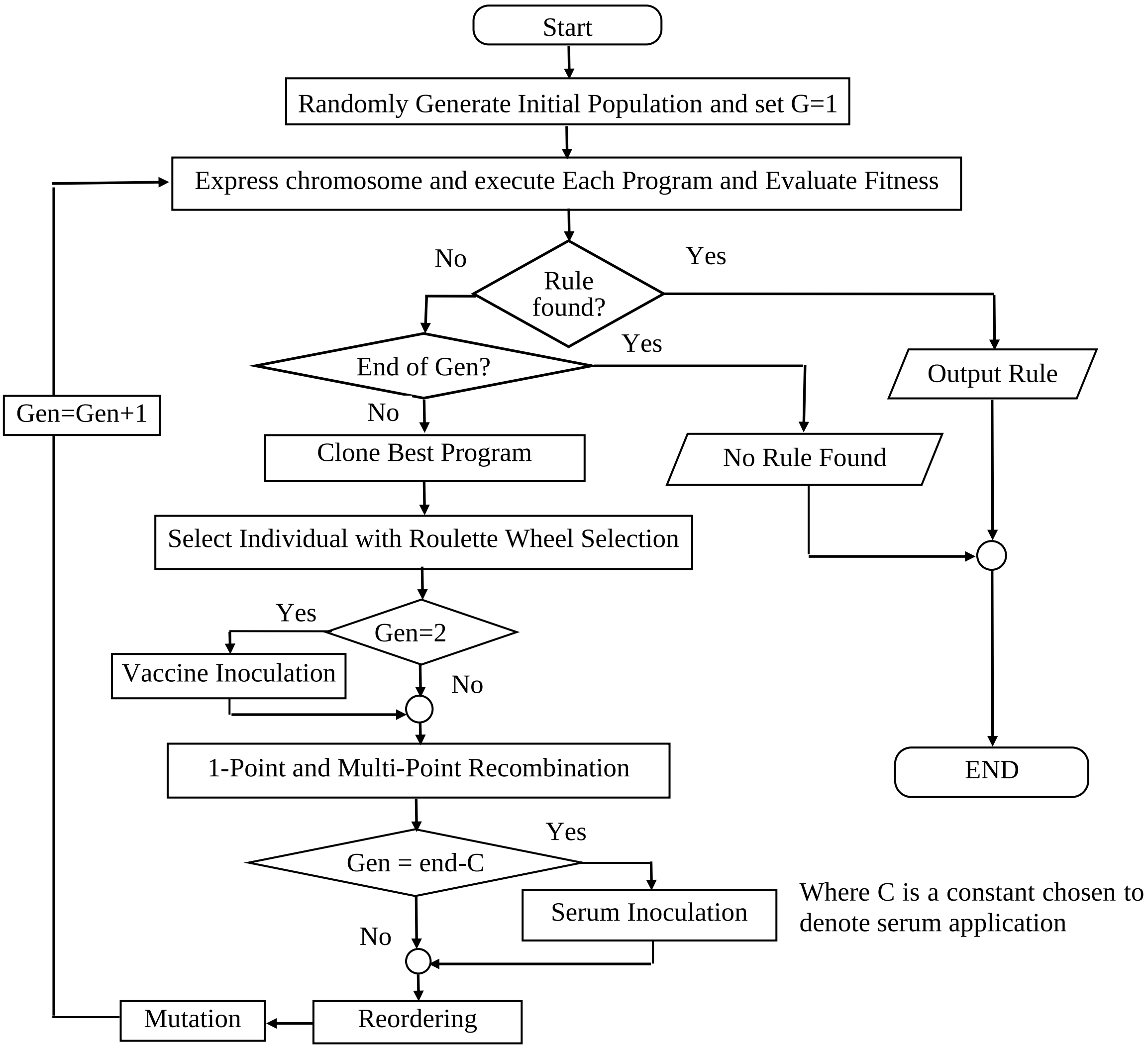

Start
Randomly Generate Initial Population and set G=1
Express chromosome and execute Each Program and Evaluate Fitness
Rule found?
No
Yes
End of Gen?
Yes
No
Output Rule
Gen=Gen+1
Clone Best Program
No Rule Found
Select Individual with Roulette Wheel Selection
Gen=2
Yes
No
Vaccine Inoculation
1-Point and Multi-Point Recombination
END
Gen = end-C
Yes
No
Serum Inoculation
Where C is a constant chosen to denote serum application
Mutation
Reordering

Figure (1) Flowchart of Controlled Gene-Accumulation Algorithm

3. Model Description

3.1. Inspiration of Nature

Returning back to nature, a chromosome is, minimally, a very long, continuous piece of DNA that contains many genes, other intervening nucleotide sequences chromosomes are genes carriers. When two genes are found on the same chromosome they are said to be linked. At the formation of gametes, when the chromosomes first line up at the mid-point of the spindle fibers, genes linked along the length of each chromosome are prone to becoming dispatched.[10] In this model, the chromosomal structure is a true simulation of nature. chromosomes are made up of several genes of different random lengths, the structure corresponds to that of the DNA in being composed of two strands that bind together and become one whole individual. Bonds used in the binding process of the strands are one-to-many relational bonds that depend entirely on the type of function to be linked.

3.2. Chromosomal Structure

In this method, the chromosome consists of a linear symbolic string of fixed total length, which is composed of one or more variable length genes that code for trees of different sizes and shapes. Each gene is composed of two strands that must bind together in order for the gene to be meaningful. The first strand contains symbols encoding the functions that perform the desired operations; it is called the F-Strand. The other strand contains symbols that encode the terminals required as operands to functions, and is called the T-Strand. Figure (2-a) gives the chromosomal structure and strands binding process, Figure (2-b) illustrates the tree expressing of genes, while Figure (2-c) depict the linking process of expression trees (ETs).

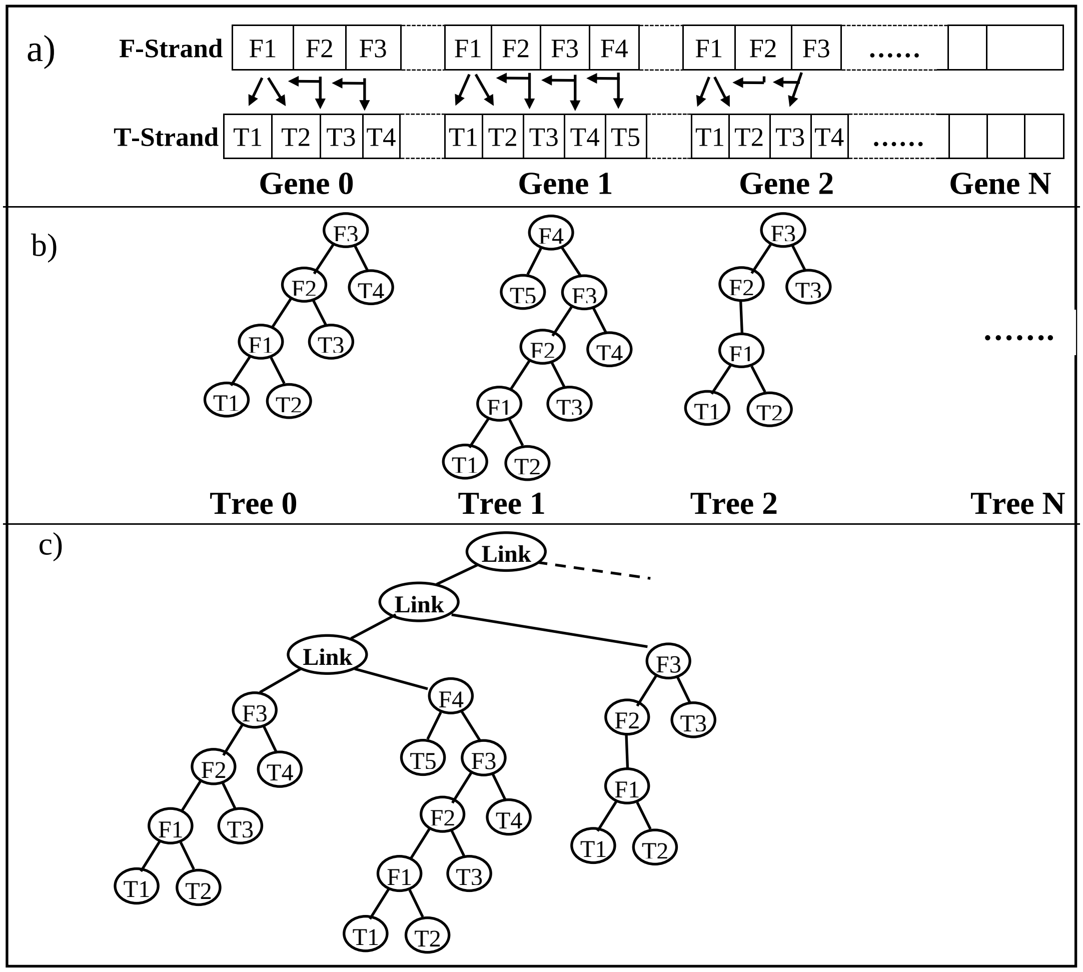


Figure (2) a) Chromosomes Structural Representation
b) Corresponding Tree Structures c) Tree Structures after linking

The Isolation of functions from terminals is very necessary to increase the efficiency and reliability in implementation and performance as:

1. It facilitates the binding process between functions and terminals.
2. It gives more efficiency in applying genetic operators with fewer constraints forced in their implementation.
3. No terminal can appear in the F-strand and stop the expansion of the open end of the corresponding expression tree.
4. Excluding terminals from the F-Strands completely eliminates the possibility of obtaining flat genes.

In F-Strands, the number of genes and their lengths is randomly chosen, forming a random length of F-Strand (LFS). The number of genes in the T-Strand is fixed accordingly to the F-strand, but the lengths of genes are not, they are instead calculated to adapt to as many as needed by the related function gene, resulting in a suitable total length of the T-Strand (LTS) that can bind correctly to the F-Strand and adapt to changes caused by operators, resulting always in correctly synthesized programs. Lengths are calculated using the following rules:

$$LTS_i = LFS_i \,.\, (MaxArg -1) +1 \qquad \ldots(1)$$

$$LTS = \Sigma_i \; LTS_i \,, \quad for\ 0 < i < N, \qquad \ldots(2)$$

Where N is the number of genes in each chromosome and *MaxArg* is the maximum number of arguments taken by a function in the function set. Thus, functions can appear in the F-Strand and takes as much terminals as needed. At the extreme case, a gene can have an F-strand of functions all requiring *MaxArg* arguments, and the whole length of the T-Strand is used. All chromosomes have the same number of genes all of the same length. Choosing the number of genes in each chromosome and their lengths is done randomly and is studied thoroughly in section (5.1)

3.3. Controlling the Gene-Accumulation Process

The main aspect of this method involves the issue of unrestricted, end-shifting utilization of chromosome length, as this is largely related to the rule complexity. This approach allows problem complexity to control the required length at evaluation time (fitness calculation), making use of just as much as needed from the chromosome. In addition, the accumulation impact assures that the components of the acquired length of the individual are actually used in the calculation.

At evaluation, as in nature, genes become dispatched. Every gene is taken in part and evaluated as an individual tree, starting with the first element in the F-Strand. Each time a branch is added, an evaluation procedure is invoked to compute the output value of the rule encoded in that tree and compare it with the desired output depending on the fitness cases chosen for training (Figure (3)). In this case there are two possibilities:

1. If no match is found, another branch is added to that tree and the process is repeated until a match is found or the end of the first gene is encountered, forming the first phenotype sub tree that is ready to be linked to the next one in the chromosome. So each gene in the chromosome is added in parts to the accumulated output value, which is controlled by continuous comparison with the desired output.
2. If a match is found, then a virtual end mark is placed to indicate the stopping end point of the chromosome. When a virtual end is found for the first fitness case, the process of tree construction ends, and the learning process begins. The newly formed rule encoded in the constructed tree is re-evaluated using the remaining fitness case in an attempt to find the fitness measure of that rule.

The total output values specified by the fitness cases must match the result stopping at the same point in the chromosome in order to state that the rule is trained and capable of solving any other new case offered.

```
For all chromosomes in the population do
    For all fitness cases in the training set do
       {
        Flag=true
        For all genes in the chromosome do
             {
             Repeat
                Add next branch to sub-tree
                Link with previously constructed tree (nil in the first gene)
                Evaluate tree
                If a match is found with the fitness case then
                    Save stopping point
                    If no match with previous stopping point then Flag=false
                    Exit repeat
            Until end of gene
            If Flag=true assign fitness according to obtained value
            Else Set Fitness=0 and Stop evaluation
            }
        Update total chromosome fitness with this fitness case
       }
```

Figure (3) Algorithm of Evaluation Process for CGAP

3.4. Genetic Operators

According to the fitness measure and the luck of the roulette wheel, individuals are selected to reproduce with modification. In CGAP, except for mutation, each operator is not allowed to modify a chromosome more than once. In addition, as in GEP, a chromosome could be chosen by one or several operators to be modified. Thus, the modifications of several genetic operators accumulate during reproduction, producing offspring very different from the parents. In CGAP, four stages of operators are introduced:

1- Mutations

Mutations can freely occur anywhere in the two-stranded chromosome, changing F-Strand’s elements to any other one defined in the function set, and T-Strand’s element into any of the terminals in the terminal set. Changes will not affect the correctness of the resulting programs.

2- Reordering

All these operators, except for Transposition, are permutation-specific, as they are guaranteed to keep all the elements but in a different reordering.

a) **Transposition**: is similar to IS and RIS transportation of GEP [5]. But here, a random sequence is chosen and is transposed to any randomly specified location in the selected chromosome.

b) **Inversion**: a random sequence is chosen from the chromosome’s length and is inverted in its place, thus reordering the genes of that sequence.

c) **Wrap-around Rotation**: a random sequence is chosen from the chromosome’s length and is rotated in its place.

d) **Flip-Folding**: the selected chromosome is divided at a predefined point into two different length parts, which are then flipped to be reunited again forming a totally new chromosome.

e) **Increment/Decrement Effect**: the contents of the chromosome are incremented or decremented by a constant creating a new chromosome.

3- Recombination

a) **One point Recombination**: the most effective operator. A point is randomly chosen in the chromosome sequence at which the contents of the two parents are swapped forming two new different children.

b) **Alternating Multi-point Recombination**: instead of using two-point crossover, a multi-point operator is invoked. It is implemented by alternately crossing over randomly chosen regions of the two selected parents. The number of points in the multi-point operator is randomly chosen and the chromosome is divided accordingly.

4- Inoculation:

This new stage involves introducing inoculation among individuals using both serums and vaccines in order to provide immunity in evolving individuals.

c) **Vaccines**: any preparation of a virus introduced into an individual to immunize it against an infection. This involves spreading a virus into the population at a very early stage of evolution, not in the first random population but in the second (first evolved population), and the system tries to settle the effect throughout the remaining generations. If individuals heal, the population grows to be immune, and if they die, the population becomes free of weak individuals.

d) **Serums**: a sequence taken from an immunized individual and used for the inoculation. At late evolution stages and after the population have reached a full-growth point and is in need of revival, the serum inoculation is used to revive such populations with vital serums.

3.5. Fitness Calculation

The most important aspect in any GP or linear GP variant method is the choice of the fitness measure for evaluating how good a given program is at solving the problem at hand. The objective must be correctly stated for the system to evolve towards discovering good solutions.

In this work, fitness functions are set accordingly to those used in GEP [4]. To evaluate the performance, GEP uses the average number of fitness-functions evaluations (Fz) needed to find a correct program with a certain probability (z). Given the success rate or the probability of success (Ps) and the number of fitness cases used in the training process (C), the number of independent runs (Rz) required to find a correct solution by generation G with a probability of z=0.99 is calculated by: [4]

$$Rz = \log(1-z) / \log(1-Ps), \quad \text{and } Ps \neq 1 \qquad ...(3)$$

$$Fz = Generations * population\_size * C * Rz \qquad ...(4)$$

4. Application to real-world problems

This approach is evaluated through its application to some of the widely referenced problems in GP and its variants. Results are compared to GEP being the most successful method of the proposed GP variants.

Tests include: Symbolic Regression, Sequence Induction, and the 11-multiplexer problems. Some of the tests use the same parameter setting of those used by GEP in order to draw some realistic conclusions. All tests carried out here use notations of success rate or probability of success (Ps) evaluated over 100 identical run. Table (1) shows the probabilities of operators for the three tests carried out in this work.

Table (1) Probabilities of Operators for Test1, 2 and 3

Operator	Test1	Test2	Test3
Mutation	0.06	0.1	0.1
Transposition	0.1	0.1	0.1
Inverse	0.1	0.15	0.1
Wrap-Around Rotate	0.3	0.35	0.1
One point recombination	0.1	0.1	0.1
Alternate Multi-point recombination	0.35	0.35	0.7
Serum	0.35	0.3	
Vaccine	0.1	0.15	

4.1. Symbolic regression

The symbolic regression problem can be stated as finding a function in a symbolic form that fits a given finite sample of data [6]. It is a valuable tool for the analysis of experimental data where the specification of the strategic function used is often difficult, and may even vary over time.[3]

As for fitness calculation, the objective here is to find an expression that performs well for all fitness cases within a certain error of the correct value. For some mathematical applications it is useful to use small relative or absolute errors in order to discover a very good solution. Mathematically, this is expressed by the following equation:

$$f_i = \sum_{j=1}^{C_i} \left(M - \left| C_{(i,j)} - T_j \right| \right) \qquad ...(5)$$

Where M is the range of selection, C(i,j) the value returned by chromosome i for fitness case j (out of Ct fitness cases) and Tj is the target value for fitness case j. If, for all j, |C(i,j) - Tj| (the precision) less or equal to 0.01, then the precision is equal to zero, and *fi* = *fmax* = Ct. M. For this problem, an M = 100 will be used, thus *fmax* = 1000. The benefit of this kind of fitness function is that the system can find optimal solutions for itself

Test1:

The first test includes applying CGAP to a symbolic regression problem; the aim is to evolve the function given in following Eq.:

$$Y = a^4 + a^3 + a^2 + a \qquad ...(6)$$

Table (2) Fitness Cases for Test1

In	2.81	6	7.043	8	10	11.38	12	14	15	20
Out	95.2425	1554	2866.5485	4680	11110	18386.0340	22620	41370	54240	168420

Probabilities of operators are given in Table (1). Fitness cases in Table (2) and parameter settings in Table (3). Being an uncomplicated function only one-gene chromosome was enough to fit the given fitness cases.

Table (3) Parameter Settings for Test1 with 100 run

Setting	CGAP	GEP
Generations	46	50
Population	30	30
Chromosome Length	19	39
Genes	1 (**F-Strand=9**)	3 (**h=6**)
Function Set	{+,-,*,/}	{+,-,*,/}
Terminal Set	{a}	{a}

CGAP Solution: *+*+*+ **/**** found in Generation 4

aaaaaaa aaa

Evaluation of the solution:

$a*a = a^2$

$+a = a^2+a$

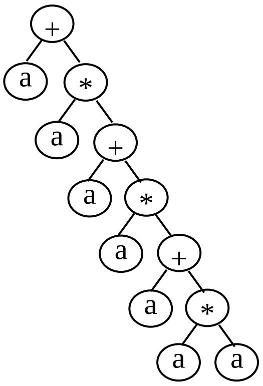
$*a = a^3+a^2$

$+a = a^3+a^2+a$

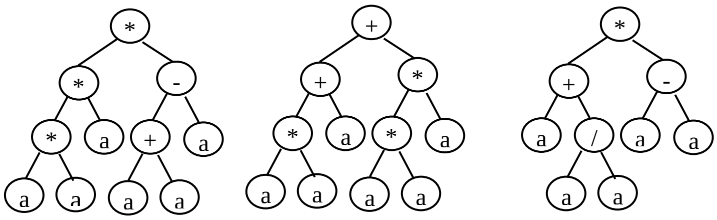
$*a = a^4+a^3+a^2$

$+a = \mathbf{a^4+a^3+a^2+a}$

PS = 1.00, Rz= 1, Fz= 13800.0000



GEP solution is **-*a+aaaaaaa ++a*aaaaaaa *+-a/aaaaaaaa**



a^4 + a^3+a^2+a + 0 = $a^4+a^3+a^2+a$

It is clear that the 3rd gene is of a zero value to the chromosome. These results indicate that CGAP surpasses GEP in many aspects. Maximum chromosomal length is significantly shorter; in addition, resulting rules are shorter and more compact. Probability of success is improved considerably with less required execution time; Table (8) gives time comparison details.

4.2. Sequence Induction

The problem of sequence induction is a special case of symbolic regression where the domain of the independent variable consists of the non-negative integers. Yet, the sequence chosen is more complicated than that used in symbolic regression, as different coefficients are used.[5]

The problem involves finding a mathematical expression to calculate the value of the sequence at a given step [9]. In the sequence 1, 15, 129, 547, 1593, 3711, 7465, 13539, 22737, 35983, 54321, …., the nth term is:

$$N = 5a_n^4 + 4a_n^3 + 3a_n^2 + 2a_n + 1, \qquad \ldots(7)$$

Where a_n consists of the non-negative integers 0, 1, 2, 3, ..., n . This sequence was chosen because it can be exactly solved and therefore can provide an accurate measure of performance in terms of success rate.

Fitness Measure for sequence induction is similar to that of Symbolic regression given in Eq. (5) for assigning fitness measure to individuals.

<u>Test2:</u>

The second test is the application of the CGAP method in sequence induction, the function to be evolved is:

$$Y = 5a^4 + 4a^3 + 3a^2 + 2a + 1 \qquad ...(8)$$

The probabilities of operators are given in Table (1), fitness cases in table (4), and parameter settings in Table (5). Gene lengths are assigned using the concave shape as will be discussed in section 5 along with other shapes in an analysis to indicate the efficiency of utilizing this shape.

Table (4) Fitness Cases for Test2

In	1	2	3	4	5	6	7	8	9	10
Out	15	129	547	1593	3711	7465	13539	22737	35983	54321

Table (5) Parameter Settings for Test2 with 100 run

Setting	**CGAP**	**GEP**
Generation	100	100
Population	60	60
Chromosomal Length	81	91
Genes	7 (**F-Strand={5,6,7,6,6,4,3}**)	7 (**h=6**)
Function Set	{+,-,*,/}	{+,-,*,/}
Terminal Set	{a}	{a}

CGAP Solution found in generation 12 with length 50, 4 genes used of maximum 85 length and 7 genes chromosomes with Linking function = ‘+’
Ps=0.79, Rz= 2.9508, Fz= 177048.3906

++***	+*+*+*	++*++*+	*+*// /	*+*/*+	+/+-	**-
aaaaaa	aaaaaaa	aaaaaaaaa	aaaaaa a			
$(3a^4)$	$(2a^4+a^3+a^2)$	$(3a^3+2a^2+a)$	$(a+1)$			

$=5a^4+4a^3+3a^2+2a+1$

Another CGAP solution of length 70 using 6 genes and Linking by ‘+’

++***	+*+**+	--+//+*	**-++/	/*++**	*+ /*	+++
aaaaaa	aaaaaaa	aaaaaaaaa	aaaaaaa	aaaaaaa	aaa aa	…
$(3a^4)$	$(2a^4+a^3+a)$	(a^2)	(a^2+1)	$(3a^3)$	(a^2+a)	

$=5a^4+4a^3+3a^2+2a+1$

GEP solution found in generation 32 with length=73 and 7 genes used of the maximum length=91 and 7 genes chromosomes, Linking function = ‘+’
Ps= 0.41, Rz= 8.7280 Fz= 523679.0625

**++—aaaaaaa	*+/+a*aaaaaaa	*+*+*+aaaaaaa	*-***+aaaaaaa	*a/+a-aaaaaaa	-+-/**aaaaaaa	**+a*+aaaaaaa

Again in this test, results of CGAP exceed GEP in utilization of chromosomal length, resulting rule size, probability of success, and in execution time. Table (8) gives comparisons in terms of execution time.

4.3. The 11-Multiplexer Problem

The goal is to evolve a rule capable of translating an input of 11 ordered binary bits into the appropriated output where the first 3 bits refer to an address of 0 to 8 and the remaining 8 bits give the input to the Multiplexer. The value of the Boolean multiplexer function is 0 or 1 of the particular data bit that is singled out by the address bits of the multiplexer. For example, if the three-address bits $a_2a_1a_0$ are 110, the multiplexer singles out data bit number 6 (d_6) to be the output of the multiplexer (Figure(4)).

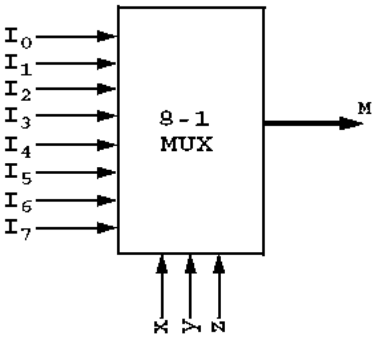


Figure (4) The 11-Multiplexer Interface

The set of fitness cases must be representative of the problem as a whole. There are 2^{11} = 2,048 possible combinations of the 11 arguments $a_0a_1a_2d_0d_1d_2d_3d_4d_5d_6d_7$ along with the associated correct value of the 11-multiplexer function. For this particular problem, sampling is used as the fitness cases for evaluating fitness. The fitness cases were assembled by address, for each address a sub-set of 20 random combinations was used each generation. Thus, a total of 160 random fitness cases were used each generation. In this case, the fitness of a rule is the number of fitness cases for which the Boolean value returned is correct, plus a bonus of 180 fitness points for each sub-set of combinations solved correctly as a whole. Hence, a total of 200 fitness points was attributed for each correctly decoded address, with 1600 as the maximum fitness. The idea is to decode one address at a time, as the individuals learn to decode first one address, then another, until the last one.

Test3:

This problem is permutation-specific, thus all the elements in the terminal set have to be present in the resulting rule in order for it to function correctly, so the operators used in this test have to be permutation-specific. In addition, a one-point recombination operator is used in a monitored manner as recombining two parents can result in two invalid offspring's that repeat or miss a certain element in the rule.

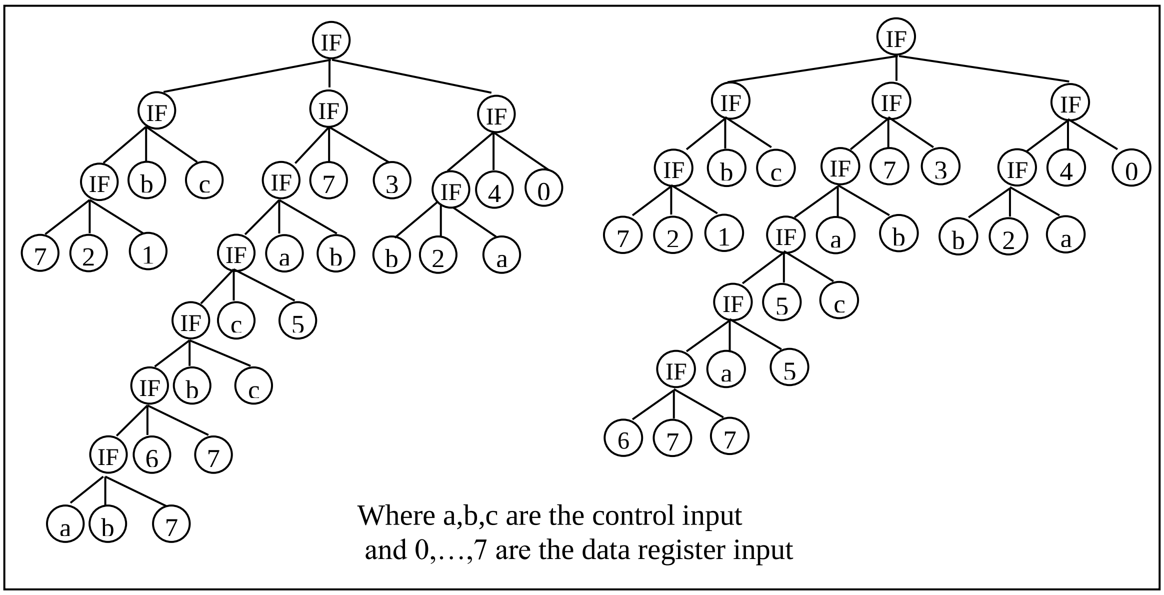
Fitness cases are randomly generated from the 2048 total possible combinations of the 11 components in a similar manner to that used by GEP. Probabilities of operators are given in Table (1) and the parameter settings in Table (6). In this test, a three-gene chromosome is used; having one element in the function set does not require an F-Strand, so a virtual one is linked with the T-Strand to form the chromosome. As a result, many different length solutions were found, listed here are two: one of length 23 with a tree of 34 nodes, and a shorter one of length 21 with a tree of 31 nodes only.

Table (6) Parameter Settings for Test3 with 100 run

Parameter	CGAP	GEP
Generation	300	400
Population	250	250
Length	21 and 23	27
Genes	3	27
Function Set	{IF}	{}
Terminal Set	{a,b,c,0,1,….,7}	{a,b,c,0,1,….,7}

Solution1: **<u>721bc</u> <u>ab767bcc5ab73</u> <u>b2a40</u>**
With 3 gene F-Strand length =10 {2,6,2}
Ps= 0.43, Rz= 8.1925, Fz= 98310232.0

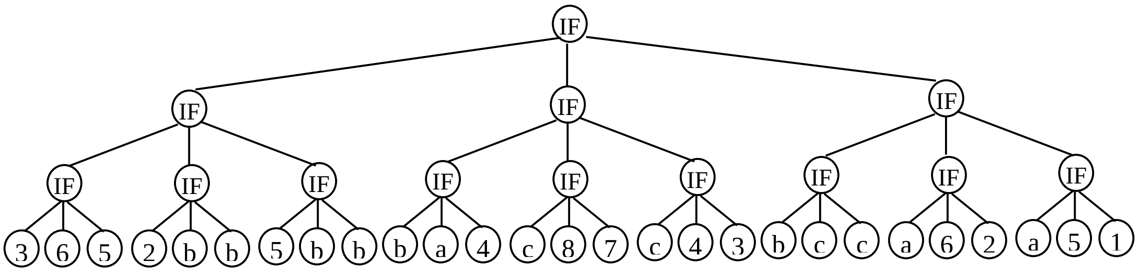
Solution2: **<u>721bc</u> <u>677a55cab73</u> <u>b2a40</u>**
With 3 gene F-Strand length = 9 {2,5,2}
Ps= 0.41, Rz= 8.7280, Fz=104735816.0



Where a,b,c are the control input
and 0,…,7 are the data register input

GEP's Solution: No function was used to generate chromosomes; the sub-ETs were linked by IF. The characters are linked 3 by 3, forming an ET with depth 4, composed of 40 nodes, the first 13 nodes are IFs, and the remaining are the chromosome characters.

<u>365</u> <u>2bb</u> <u>5bb</u> <u>ba4</u> <u>c87</u> <u>c43</u> <u>bcc</u> <u>a62</u> <u>a51</u>
Ps= 0.32, Rz= 11.9409, Fz= 191054944.0


IF
IF IF IF
IF IF IF IF IF IF IF IF IF
3 6 5 2 b b 5 b b b a 4 c 8 7 c 4 3 b c c a 6 2 a 5 1

The examination performed in the course of this test showed that results obtained by using CGAP technique outperform GEP in chromosome length, output rule size, probability of success, and in execution time, as indicated in Table (8).

5. Comparisons between CGAP and GEP:

5.1. Chromosome Size and Gene length:

In GEP, the chromosome length is increased gradually until the suitable length is found, if no acceptable success rate is gained, then the length for the head is fixed and the number of genes is increased. With the help of the linking function, more compound tree structures are constructed. Because GEP uses fixed length genes, and a fixed total chromosome length, i.e., all genes are used in finding the fitness of an individual, the choice of gene length and number of genes in the chromosome is very sensitive and can dangerously influence the evolutionary process.

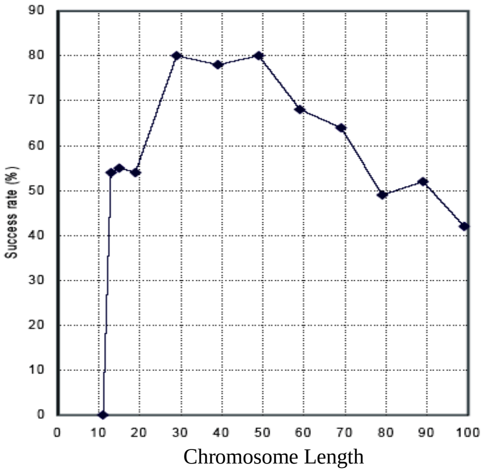


Figure (5) Effect of increasing Chromosome Length with Success rates in GEP

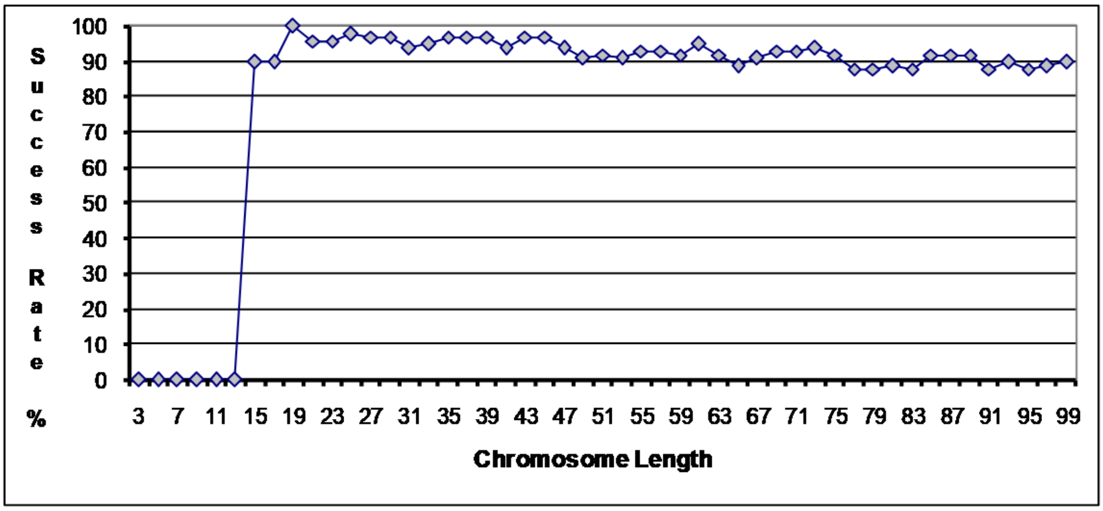


Figure (6) Effect of Increasing Chromosome Length with Success Rates in CGAP

The relationship between success rate and chromosomal length is therefore very complicated. Success rates continue to increase with the length to a certain point, after that any further increase will only decrease success rates as stated in Figure (5) [4, 7]. In CGAP the situation is considerably different; having genes of different lengths adds more flexibility. In addition, the open varying end of the chromosome reduces the impact of total length on success rates. In this section, chromosome length is investigated and compared to GEP. The investigation was done using the environment of single-gene chromosome population stated in Test1 (which is the same test used in Figure (5)); length was varied from 0 to 100 with a population of 46 individual. Results are given in Figure (6). Even with very long chromosomes (length of 100) success rates are still very close to that obtained with short reasonable length, and are very high relatively to GEP.

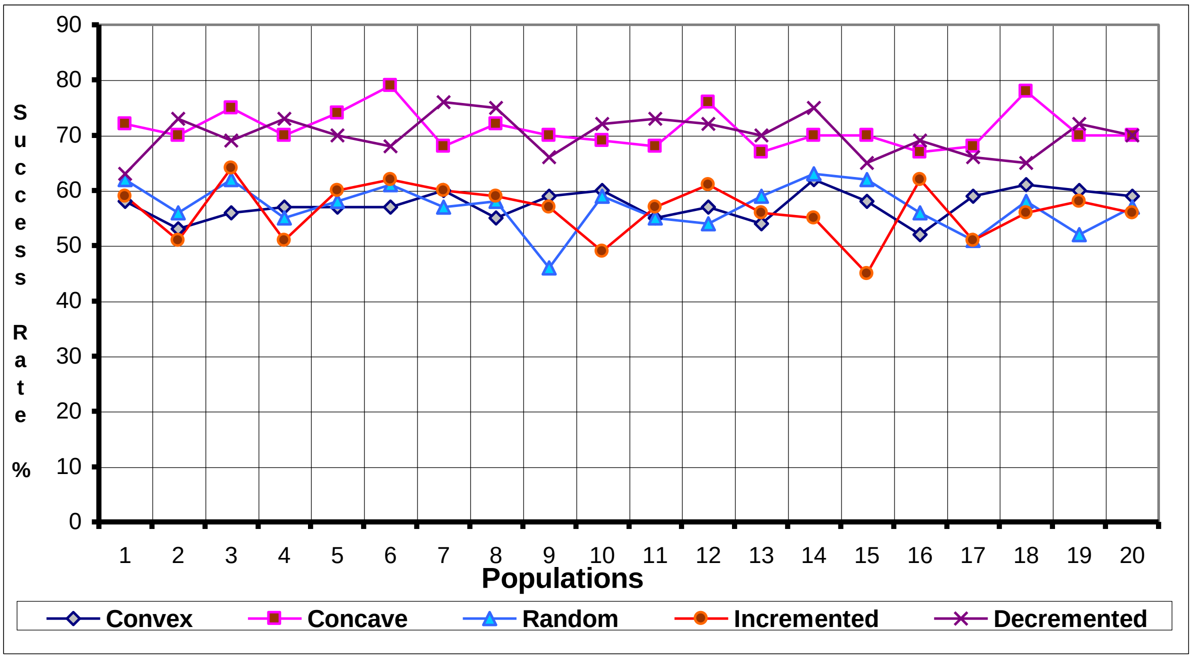


Figure (7) Relationship between Chromosomal Shapes and Success Rates

As for multi-gene chromosome populations, Sequence Induction with a population of 60 individuals was used to find the finest possible layout or shape for assigning lengths to genes in the chromosomes. The number of genes is randomly chosen, and the lengths of genes should be around the average value obtained by dividing total chromosomal length by the number of genes, environmental settings are identical to those stated in Test2. Empirical testing shows that the best results can be obtained when assigning short-length genes at both ends and increasing length as proceeding towards the middle, giving the chromosome a Concave form. Different arrangements of this shape, like: {5,6,7,7,6,4,3} or {4,6,7,5,3}, provide a gradual increase in expression complexity, after reaching the chromosome center, the process is reversed to supply less complex terms at the end of final expressions.

Figure (7) shows success rates of five different shapes for chromosome gene lengths, a total of 20 populations of the same size and environment as indicated in Test2 are used; total chromosome length is equal to 81 and is composed of 7 genes. Each of these 20 populations employs 60 individual of different arrangement for gene lengths all following the same shape of the total five chromosome shapes as follows:

1. **Convex shape** decreasing gene lengths towards the middle.
2. **Concave shape** increasing gene lengths towards the middle.
3. **Random shape** of no particular style. it can randomly produce any shape.
4. **Incremented shape** continually increasing gene lengths towards the end.
5. **Decremented shape** continually decreasing gene lengths towards the end.

Results indicate that Concave-shaped and Decremented-shaped chromosomes gave best success rates among all others. Table (7) ranks these shapes by their average success rates.

Table (7) Ranking of Chromosome Shapes by Average Success Rates

Rank	Shape	Average Success Rate
1	Concave Shape	71.15
2	Decremented Shape	70.10
3	Convex Shape	57.45
4	Random Shape	57.05
5	Incremented Shape	56.45

5.2. Comparisons of Results by Execution Time

For a complete test satisfaction, the computational time for executing runs in all tests was measured and compared with GEP. Each test is done using a Pentium (300MHz) processor. Table (8) shows the results in terms of success rates and execution time.

Table (8) Comparisons of Execution Time and Ps for 100 Run

	CGAP		GEP	
	Time*	**Ps**	**Time***	**Ps**
Test1	00:00:05:82	1.0	00:00:08:36	0.81
Test2	00:01:46:66	0.79	00:03:11:24	0.41
Test3	02:20:50:00	0.43	08:40:00:00	0.32

Time is expressed as {Hour:Minute:Second:Hundredth of second}

Due to the fact that successful runs end earlier than unsuccessful ones, the same test was carried out to measure the time used to complete one unsuccessful run. Test1 was done using 200 generations in order to make the amount of time significant. Reasonably, the time shown in Table (9) is an average of a total of 20 evaluated runs.

Test3 was executed on a faster computer of (2.40GHz) processor and the time measured was {00:27:23:41}

Table (9) Comparisons of Execution Time for One Unsuccessful Run

	CGAP Time	GEP Time
Test1	00:00:00:21	00:00:00:43
Test2	00:00:01:64	00:00:02:40
Test3	00:01:10:22	00:05:27:12

Table (10) Comparison between CGAP and GEP

	CGAP	GEP
1.	Controlled length of chromosome that adapts to rule complexity.	Required length is found by tests relying on rule complexity.
2.	Easy evaluation of fitness function, results is just accumulated.	Complicated, Karva language is used.
3.	Short efficient resulting programs.	Fixed relatively large less efficient programs.
4.	Additional non-coding regions are kept at end and neglected in results	Non-coding regions spread in the rule
5.	Genes of different lengths allow more flexibility.	Use of multigenic families of equal length restricts the flexibility
6.	Use of inoculation in utilizing operators with its immunity feature	Use of mutation, transposition and recombination only. No inoculation operator.
7.	No flat genes	Flat genes are very common
8.	No illegal operations. At linking, it may only occur with divisions.	Illegal operations in genes are very common.
9.	Shorter execution time	Evaluating all genes requires longer execution time
10.	Requires less generations	Requires more generations
11.	Increasing chromosome length does not significantly affect success rates	Increasing chromosome length affect success rates considerably.

Adding inoculation as a new stage to genetic operators has imposed a massive influence on the behavior of the system. Both inoculations were inspired by nature and proved to be successful in applications.

As a final conclusion on the comparisons made in this section, Table (10) reviews the important issues in which CGAP is found superior to GEP.

6. Conclusions and further recommendations:

Based on ideas motivated by nature, a new evolutionary method was proposed to solve real-world problems in an automated way. *Controlled Gene Accumulation Programming* was developed aiming to overcome malfunctioning phenomena of other existing methods. The newly developed method presented proved to be efficient and reliable in problem solving in terms of chromosome size, success rates and execution time.

The new model was successfully applied to various benchmark problems: symbolic regression, sequence induction, and the 11-Multiplexer problem. It was investigated and compared to prove superiority over Gene expression Programming.

In spite of the tremendous work that has been done in the field of Evolutionary Algorithms and Automatic programming, there are still many issues that might arise in the context of research relating to problem solving.

As for further recommendations, CGAP being newly introduced can be used in different areas of applications to show its impact on finding satisfactory rules to complicated problems.

REFERENCES

[1] AL-Saati, N.A., (2005), A Novel Proposed Model for Automatic Programming in Problem Solving, PhD Thesis, University of Mosul, College of Computers and Mathematical Sciences, Mosul, Iraq, 147p.

[2] Banzhaf, W., (2001), A Comparison of linear Genetic Programming and neural Networks in medical Data Mining, IEEE Transactions on Evolutionary Computation, Vol. 5(1), pp: 17-26.

[3] Duffy, J., and Engle-Warnick, J., (2002), Using Symbolic Regression to Infer Strategies from Experimental Data. In S-H Chen, Ed., Evolutionary computation in economics and finance New-York Physica-Verlag.

[4] Ferreira, C., (2001), Gene Expression Programming: A new Adaptive Algorithm for Solving Problems, in Complex Systems,13(2),pp:87-129.

[5] Ferreira, C., (2002), Gene Expression Programming in Problem Solving, In R. Roy, M. Köppen, S. Ovaska, T. Furuhashi, and F. Hoffmann, eds., Soft Computing and Industry – Recent Applications, pages 635-654, Springer-Verlag, 2002.

[6] Hoai, N.X., (2001), Solving the Symbolic Regression with Tree-Adjunct Grammar Guided Genetic Programming: The Preliminary Results, In The Proceedings of The 5th Australasia-Japan Joint Workshop on Evolutionary Computation and Intelligent Systems (AJWIES), Dunedin, New Zealand,19-21st Nov. 2001, pp:1-6.

[7] Oltean M., Dumitrescu D., (2002), Multi Expression Programming, Technical Report: UBB-01-2002, Babes-Bolyai University, Cluj-Napoca, Romania, in Journal of Genetic Programming and Evolvable Machines, Kluwer, second tour of review, 33p.

[8] Ryan, C., Collins, J.J., O'Neill, M., (1998), Grammatical Evolution: Evolving Programs for an Arbitrary Language, Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming, Springer-Verlag pp: 83-95.

[9] Wang, J-S., (2003), Influences of Function Sets in Genetic Programming, in Genetic Algorithms and Genetic Programming at Stanford 2003, Ed. John R. Koza, Stanford Bookstore Publisher, pp:221-229.

[10] Wikipedia: A Web-based free content encyclopedia, cited at:(*http://en*. wikipedia.org/wiki/Wikipedia)