Parallel Grammatical Evolution

Pavel Ošmera,
Institute of Automation and Computer Science
Brno University of Technology
Faculty of Mechanical Engineering
Brno, Czech Republic
osmera@fme.vutbr.cz

Tomáš Panáček,
Institute of Automation and Computer Science
Brno University of Technology
Faculty of Mechanical Engineering
Brno, Czech Republic
panacek@ro.vutbr.cz

Petr Pivoňka
Department of Control and Instrumentation
Brno University of Technology
Faculty of Electrical Engineering and Communication
Brno, Czech Republic
pivonka@feec.vutbr.cz

ABSTRACT

This paper describes Parallel Grammatical Evolution (PGE) that can evolve complete programs using a variable length linear genome to govern the mapping of a Backus Naur Form grammar definition. To increase the efficiency of Grammatical Evolution (GE) the influence of backward processing was tested. The significance of backward coding (BC) and the comparison with standard coding of GEs is presented. BC can speed up Grammatical Evolution with high quality features. The adaptive significance of Parallel Grammatical Evolution with "male" and "female" populations has been studied.

Categories and Subject Descriptors

D.2.2 Delphi

Index Terms

Algorithms, Experimentation, Theory.

Keywords

grammatical evolution, backward processing, parallel evolution.

I. INTRODUCTION

Grammatical Evolution (GE) [1] can be considered a form of grammar-based genetic programming (GP). In particular, Koza's genetic programming has enjoyed considerable popularity and widespread use. Unlike a Koza-style approach, there is no distinction made at this stage between what he describes as function (operator in this case) and terminals (variables). Koza originally employed Lisp as his target language. This distinction is more of an implementation detail than a design issue. Grammatical evolution can be used to generate programs in any language, using Backus Naur Form (BNF). BNF grammars consist of terminals, which are items that can appear in the language, i.e. +, -, sin, log etc. and non-terminal, which can be expanded into one or more terminals and non-terminals. A non-terminal symbol is any symbol that can be rewritten to another string, and conversely a terminal symbol is one that cannot be rewritten. The major strength of GE with respect to GP is its ability to generate multi-line functions in any language. Rather than representing the programs as parse tree, as in GP, a linear genome representing is used [1]-[3]. A genotype-phenotype mapping is employed such that each individual's variable length byte strings, contains the information to select production rules from a BNF grammar. The grammar allows the generation of programs, in an arbitrary language that are guaranteed to be syntactically correct. The user can tailor the grammar to produce solutions that are purely syntactically constrained, or they may incorporate domain knowledge by biasing the grammar to produce very specific form of sentences.

Because, GE mapping technique employs a BNF definition, the system is language independent, and theoretically can generate arbitrarily complex functions. There is quite an unusual approach in GEs, as it is possible for certain genes to be used two or more times if the wrapping operator is used. BNF is a notation that represents a language in the form of production rules. It is possible to generate programs using the Grammatical Swarm Optimization (GSO) technique [2] with a performance similar to the GE. The relative simplicity, the small population sizes, and the complete absence of a crossover operator synonymous with program evolution in GP or GE are main advantages of GSO. Grammatical evolution was one of the first approaches to distinguish between the genotype and phenotype. GE evolves a sequence of rule numbers that are translated, using a predetermined grammar set into a phenotypic tree.

Our approach uses a parallel structure of GE (PGE). A population is divided into several sub-populations that are arranged in the hierarchical structure [4]. Every sub-population has two separate parts: a "male" group and a "female" group. Every group uses quite a different type of selection. In the first group a classical type of GA selection is used. To the second group only different individuals can be added. This strategy was inspired by harem system in Nature that solves problem of adaptation complex organisms to microorganisms [8], [9]. The biologically inspired strategy increases an inner adaptation of PGE. This analogy would lead us one step further, namely, to the belief that the combination of GE with 2 different selections that are simultaneously used can improve an adaptive behavior of GE [5] - [7]. On the principle of two selections we can create a parallel GE with a hierarchical structure (see Fig. 9).

II. PARALLEL GRAMMATICAL EVOLUTION

The PGE is based on the grammatical evolution GE [1], where BNF grammars consist of terminals and non-terminals. Terminals are items, which can appear in the language. Non-terminals can be expanded into one or more terminals and non-terminals. Grammar is represented by the tuple {N,T,P,S}, where N is the set of non-terminals, T the set of terminals, P a set of production rules which map the elements of N to T, and S is a start symbol which is a member of N. For example, below is the BNF used for our problem:

N = {expr, fnc}
T = {sin, cos, +, -, /, *, X, 1, 2, 3, 4, 5, 6, 7, 8, 9}
S = <expr>
and P can be represented as 4 production rules:
1. <expr> := <fnc><expr>
<fnc><expr><expr>
<fnc><num><expr>
<var>
2. <fnc> := sin
cos
+
*
-
U-
3. <var> := X
4. <num> := 0,1,2,3,4,5,6,7,8,9

The production rules and the number of choices associated with each are in Table 1. The symbol U- denotes an unary minus operation.

Table 1: The number of available choices for an each production rule

rule no	choices
1	4
2	6
3	1
4	10

There are notable differences when compared with [1]. We don't use two elements <pre_op> and <op>, but only one element <fnc> for all functions with n arguments. There are not rules for parentheses; they are substituted by a tree representation of the function. The element <num> and the rule <fnc><num><expr> were added to cover generating numbers. The rule <fnc><num><expr> is derived from the rule <fnc><expr><expr>. Using this approach we can generate the expressions more easily. For example when one argument is a number, then +(4,x) can be produced, which is equivalent to (4 + x) in an infix notation. The same result can be received if one of <expr> in the rule <fnc><expr><expr> is substituted with <var> and then with a number, but it would need more genes.

There are not any rules with parentheses because all information is included in the tree representation of an individual. Parentheses are automatically added during the creation of the text output.

If in the GE is not restricted anyhow, the search space can have infinite number of solutions. For example the function cos(2x), can be expressed as cos(x+x); cos(x+x+1-1); cos(x+x+x-x); cos(x+x+0+0+0...) etc. It is desired to limit the number of elements in the expression and the number of repetitions of the same terminals and non-terminals.

III. BACKWARD PROCESSING OF THE GE

The chromosome is represented by a set of integers filled with random values in the initial population. Gene values are used during chromosome translation to decide which terminal or nonterminal to pick from the set. When selecting a production rule there are four possibilities, we use gene_value mod 4 to select a rule. However the list of variables has only one member (variable X) and gene_value mod 1 always returns 0. A gene is always read; no matter if a decision is to be made, this approach makes some genes in the chromosome somehow redundant. Values of such genes can be randomly created, but genes must be present.

The figure Fig. 1 shows the genotype-phenotype translation scheme. Body of the individual is shown as a linear structure, but in fact it is stored as a one-way tree (child objects have no links to parent objects). In the diagram we use abbreviated notations for nonterminal symbols: f - <fnc>, e - <expr>, n - <num>, v - <var>.

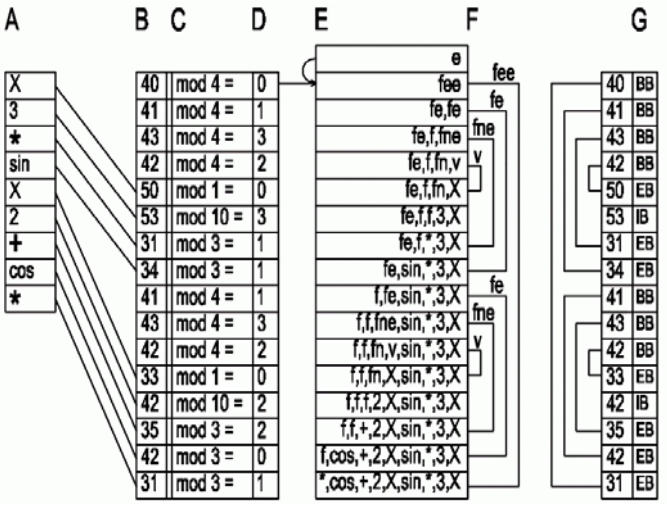

Figure 1. Relations between genotype (column B) and phenotype (column A)

The column description in Fig. 1:

A. Objects of the individual's body (resulting trigonometric function),
B. Genes used to translate the chromosome into the phenotype,
C. Modulo operation, divisor is the number of possible choices determined by the gene context,
D. Result of the modulo operation,
E. State of the individual's body after processing a gene on the corresponding line,
F. Blocks in the chromosome and corresponding production rules,
G. Block marks added to the chromosome.

Since operation modulo takes two operands, the resulting number is influenced by gene value and by gene context (Fig. 1C = see Fig. 1 column C). Gene context is the number of choices, determined by the currently used list (rules, functions, variables). Therefore genes with same values might give different results of modulo operation depending on what object they code. On the other hand one terminal symbol can be coded by many different gene values as long as the result of modulo operation is the same (31 mod 3) = (34 mod 3) = 1. In the example (Fig. 1A) given the variables set has only one member X. Therefore, modulo divider is always 1 and the result is always 0, a gene which codes a variable is redundant in that context (Fig. 1D). If

the system runs out of genes during phenotype-genotype translation then the chromosome is wrapped and genes at the beginning are reused.

IV. PROCESSING THE GRAMMAR

The processing of the production rules is done backwards – from the end of the rule to the beginning (Fig. 2). E.g. production rule <fnc><expr1><expr2> is processed as <expr2><expr1><fnc>. We use <expr1> and <expr2> at this point to denote which expression will be the first argument of <fnc>.

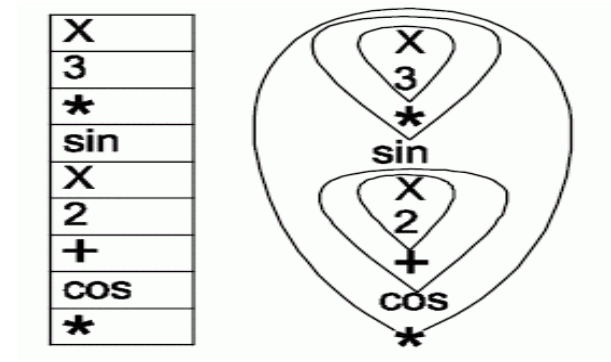

Figure 2. Proposed backward notation of a function tree structure

The main difference between <fnc> and <expr> nonterminals is in the number of real objects they produce in the individual's body. Nonterminal <fnc> always generates one and only one terminal; on the contrary <expr> generates an unknown number of nonterminal and terminal symbols. If the phenotype is represented as a tree structure then a product of the <fnc> nonterminal is the parent object for handling all objects generated by <expr> nonterminals contained in the same rule. Therefore the rule <fnc><expr1><expr2> can be represented as a tree (Fig. 3).

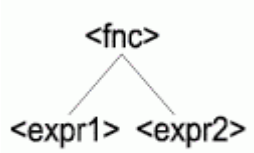

Figure 3. Production rule shown as a tree

To select a production rule (selection of a tree structure) only one gene is needed. To process the selected rule a number of n genes are needed and finally to select a specific nonterminal symbol again one gene is needed. If the processing is done backwards the first processed terminals are leafs of the tree and the last processed terminal in a rule is the root of a subtree. The very last terminal is the root of the whole tree. Note that in a forward processing (<fnc><expr1><expr2>) the first processed gene codes the rule, the second gene codes the root of the subtree and the last are leafs.

When using the forward processing and coding of the rules described in [1] it's not possible to easily recover the tree structure from genotype. This is caused with <expr> nonterminals using an unknown number of successive genes. The last processed terminal being just a leaf of the tree. The proposed backward processing is shown in Fig. 1E.

4.1 Phenotype to genotype projection

Using the proposed backward processing system the translation to a phenotype subtree has a certain scheme. It begins with a production rule (selecting the type of the subtree) and ends with the root of the subtree (in our case with a function) (Fig. 1F). In the genotype this means that one gene used to select a production rule is followed by n genes with different contexts which are followed by one gene used to translate <fnc>. Therefore a gene coding a production rule forms a pair with a gene coding terminal symbol for <fnc> (root of the rule). Those genes can be marked when processing the individual. This is an example of a simple marking system:

BB – Begin block (a gene coding a production rule)
IB – Inside block
EB – End block (a gene coding a root of a subtree)

The EB and BB marks are pair marks and in the chromosome they define a block (Fig. 1G). Such blocks can be nested but they don't overlap (the same way as parentheses). The IB mark is not a pair mark, but it is always contained in a block (IB marks are presently generated by <num> nonterminals). Given a BB gene a corresponding EB gene can be found using a simple LIFO method.

A block of chromosome enclosed in a BB-EB gene pair then codes a subtree of the phenotype. Such block is fully autonomous and can be exchanged with any other block or it can serve as completely new individual.

Only BB genes code the tree of individual's body, while EB and IB genes code the terminal symbols in the resulting phenotype. The BB genes code the structure of the individual, changing their values can cause change of the applied production rule. Therefore change (e.g. by mutation) in the value of a structural gene may trigger change of context of many, or all following genes.

This simple marking system introduces a phenotype feedback to phenotype; however it doesn't affect the universality of the algorithm. It's not dependent on the used terminal or nonterminal symbols; it only requires the result to be a tree structure. Using this system it's possible to introduce a progressive crossover and mutation.

4.2 Crossover

When using grammatical evolution the resulting phenotype coded by one gene depends on the value of the gene and on its context. If a chromosome is crossed at random point, it is very probable that the context of the genes in second part will change. This way crossover causes destruction of the phenotype, because the newly added parts code different phenotype than in the original individual.

This behavior can be eliminated using a block marking system. Crossover is then performed as an exchange of blocks. The crossover is made always in an even number of genes, where the odd gene must be BB gene and even must be EB gene. Starting BB gene is presently chosen randomly; the first gene is excluded because it encapsulates (together with the last used gene) the whole individual.

The operation takes two parent chromosomes and the result is always two child chromosomes. It is also possible to combine the same individuals, while the resulting child chromosomes can be entirely different.

Given the parents:

1) cos(x + 2) + sin(x * 3)
2) cos(x + 2) + sin(x * 3)

The operation can produce children:

3) cos(sin(x * 3) + 2) + sin(x * 3)
4) cos(x + 2) + x

This crossover method works similar to direct combining of phenotype trees, however this method works purely on the chromosome. Therefore phenotype and genotype are still separated. The result is a chromosome, which will generate an individual with a structure combined from its parents. This way we receive the encoding of an individual without backward analysis of his phenotype. To perform a crossover the phenotype has to be evaluated (to mark the genes), but it is neither used nor know in the crossover operation (also it doesn't have to exist).

4.3 Mutation

Mutation can be divided into mutation of structural (BB) genes and mutation of other genes. Mutation of one structural gene can affect other genes by changing their context therefore structural mutation amount should be very low. On the other hand the amount of mutation of other genes can be set very high and it can speed up searching an approximate solution.

Given an individual:

sin(2 + x) + cos(3 * x)

and using only mutation of non-structural genes, it is possible to get:

cos(5 – x) * sin(1 * x)

Therefore the structure doesn't change, but we can get a lot of new combinations of terminal symbols. The divided mutation allows using the benefits of high mutation while eliminating the risk of damaging the structure of an individual.

4.4 Population model

The system uses three populations forming a simple tree structure (Fig. 4). There is a Master population and two slave populations, which simulate different genders. The links among the populations lead only one way - from bottom to top.

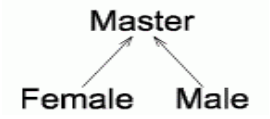

Figure 4. The population model

4.5 Female population

When a new individual is to be inserted in a population a check is preformed whether it should be inserted. If a same or similar individual already exists in the population then the new individual is not inserted. In a female population every genotype and phenotype occurs only once. The population maintains a very high diversity; therefore the mutation operation is not applied to this population. Removing the individuals is based on two criterions. The first criterion is the age of an individual - length of stay in the population. The second criterion is the fitness of an individual. Using the second criterion a maximum population size is maintained. Parents are chosen using the tournament system selection.

4.6 Male population

New individuals are not checked so duplicate phenotypes and genotypes can occur, also the mutation is enabled for this population. Mutation rate can be safely set very high (30%) provided that the structural mutation is set very low (less then 2%). For a couple of best individuals the mutations are nondestructive. If a protected individual is to be mutated a clone is created and added to the population. If the system stagnates in a local solution the mutation rate is raised using a linear function depending on the number of cycles for which the solution wasn't improved. Parents are chosen using a logarithmic function depending on the position of an individual in a population sorted by fitness. For every selected male parent a new selection of female parent is made.

4.7 Master population

The master population is superior to the male and female populations. Periodically the subpopulations send over their best solutions. Moreover the master population performs another evolution on its own. Parents are selected using the tournament system. The master population uses the same system of mutations as the male population, but for removing individuals from the population only the fitness criterion is used. Therefore master population also serves as an archive of best solutions of the whole system.

4.8 Fitness function

Around the searched function there is defined an equidistant area of a given size. Fitness of an individual's phenotype is computed as the number of points inside this area divided by the number of all checked points (a value in <0,1>). This fitness function forms a strong selection pressure; therefore the system finds an approximate solution very quickly.

4.9 Results

Given sample of 100 points in the interval $[0,2\pi]$ and using the block marking system described in 4.1, PGE has successfully found the searched function sin (2*x)*cos (2+x) on the majority of runs. The system also found a function with fitness better then 0.8 during less than 40 generations using merely 3 populations (M, F, MR) with size = 100 individuals in each (Fig. 5).

The graph (Fig 6.) shows maximum fitness in the system for ten runs and an average (bold). On the other hand the same system with phenotype to genotype projection disabled (Fig. 6). The majority of runs didn't find the searched function within 120 generations.

We have simplified the generation of numbers by adding a new production rule, thus allowing the generation of functions containing integer constants. The described parallel system together with phenotype to genotype projection improved the speed of the system. The progressive crossover and mutation eliminates destroying partial results and allowed us to generate more complicated functions (e.g. sin(2 * x)*cos(2 + x)).

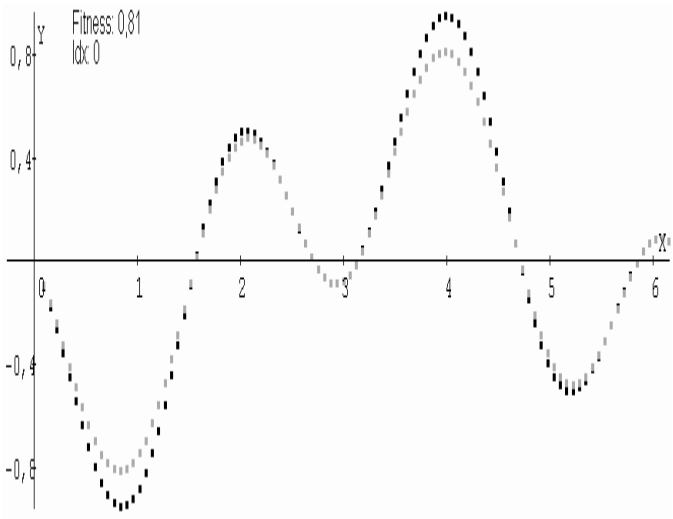

Figure 5. A generated function with fitness 0.81 *-sin(sin(-9 + x)) * cos(-11 + 2*x)* (light) and searched function *sin(2 * x)*cos(2 + x)* (bold)

We have described a parallel system, Parallel Grammatical Evolution (PGE) that can map an integer genotype onto a phenotype with the backward coding. PGE has proved successful for creating trigonometric identities.

Parallel GEs with the sexual reproduction can increase the efficiency and robustness of systems, and thus they can track better optimal parameters in a changing environment. From the experimental session it can be concluded that modified standard GEs with two sub-populations can design PGE much better than classical versions of GEs.

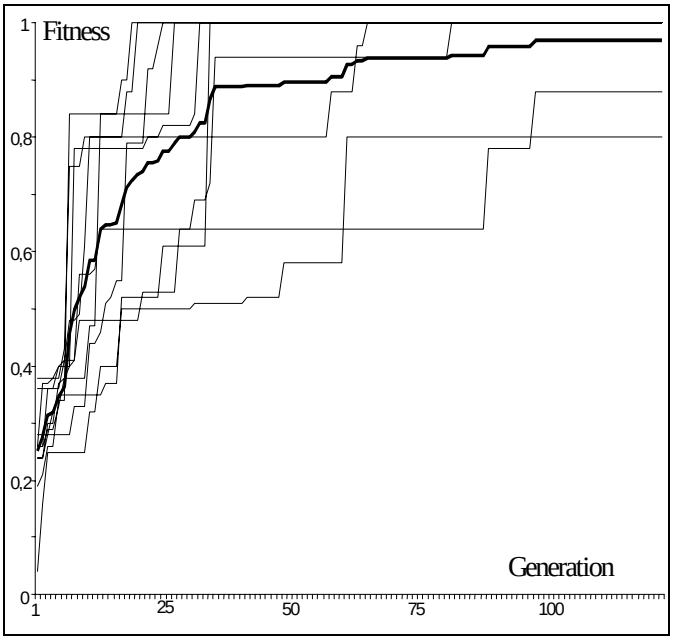

Figure 6. Convergence of the PGE using backward processing (average in bold)

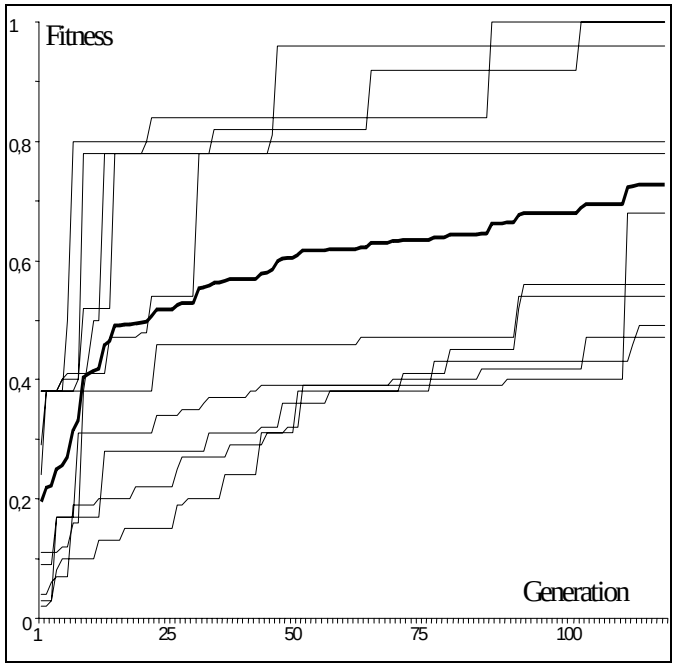

Figure 7. Convergence of the PGE using forward processing (average in bold)

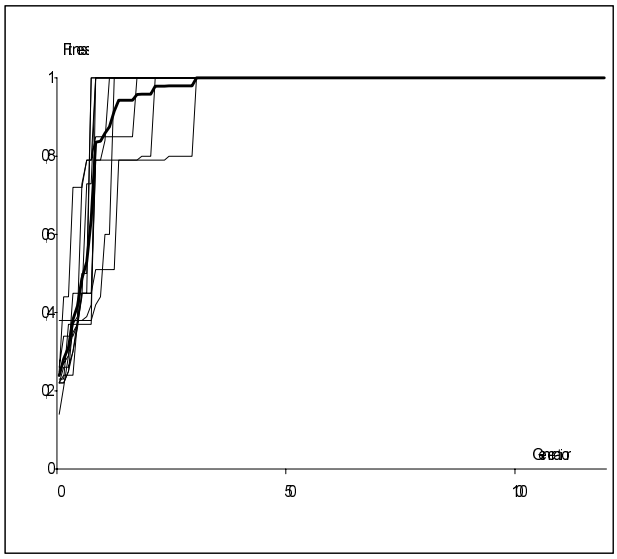

Figure 8. Convergence of the PGE with 5 PC using backward processing (average in bold)

The PGE algorithm was tested with the group of 6 computers in the computer network (see Fig. 9). Five computers calculated in the structure of five subsystems MR1, MR2, MR3, MR4, and MR5 and one master MR. The male subpopulation M of MR in the higher level follows the convergence of the subsystem. In Fig. 8 is presented 10 runs of the PGE- program. The shortest time of computation is only 10 generation. All calculation were finished before 40 generation. This is better to compare with backward processing on one computer (see Fig. 6). The forward processing on one computer was the slowest (see Fig. 7).

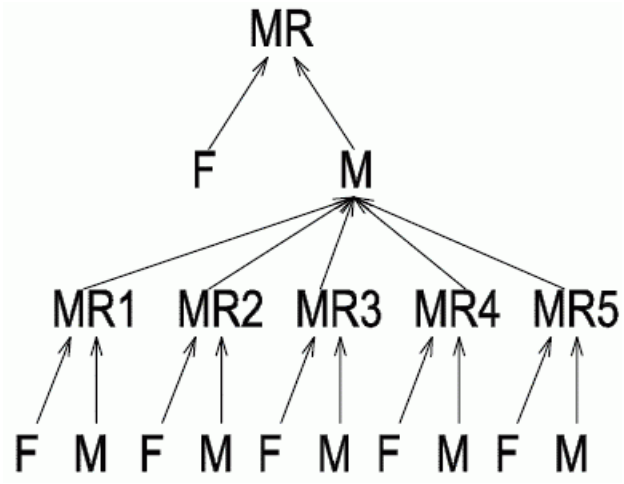

Figure 9. The parallel structure of PGE with 6 computers

4.10 Logical function XOR

Input values are two integer numbers a and b; a, b 2< 0, 1 >. Output number c is the value of logical function XOR. Training data is a set of triples (a, b, c):

P = {(0, 0, 0); (0, 1, 1); (1, 0, 1); (1, 1,0)}.

Thus the training set represents the truth table of the XOR function. The function can be expressed using _, ^, ¬ functions:

a + b = (a ^ ¬b) _ (¬a ^ b) = (a _ b) ^ (¬a _ ¬b) = (a _ b) ^ ¬(a ^ b)

The grammar was simplified so that it does not contain conditional statement and numeric constants, on the other hand three new terminals were added to generate functions _, ^, ¬. Thus the grammar generates representations of the XOR functions using other logical functions.

1)

```
function xxor($a,$b) {
$result = "no_value";
$result = ($result) |
((((~(~(~(~(~($result)))))) | (($a) & (($a) & (~($b))))) & ($a))
| ((~($a)) & ($b)));
return $result;
}
```

Number of generations: 32

2)

```
function xxor($a,$b) {
$result = "no_value";
$result = ($result) | (((~$b & ($a & ($a & ~$b))) & $a) |
(~$a & $b));
return $result;
```

Number of generations: 53

V. CONCLUSION

We have simplified the generation of numbers by adding a new production rule, thus allowing the generation of functions containing integer constants. The described parallel system together with phenotype to genotype projection improved the speed of the system. The progressive crossover and mutation eliminates destroying partial results and allowed us to generate more complicated functions (e.g. sin(2 * x)*cos(2 + x)).

We have described the Parallel Grammatical Evolution (PGE) that can map an integer genotype onto a phenotype with the backward processing. PGE has proved successful for creating trigonometric functions.

Parallel GEs with hierarchical structure can increase the efficiency and robustness of systems, and thus they can track better optimal parameters in a changing environment. From the experimental session it can be concluded that modified standard GEs with only two sub-populations can create PGE much better than classical versions of GEs.

The parallel grammatical evolution can be used for the automatic generation of programs. We are far from supposing that all difficulties are removed but first results with PGEs are very promising.

ACKNOWLEDGMENTS

This work has been supported by Czech Grant Agency grant No: 102/06/1132 Soft Computing in Control.

REFERENCES

[1] O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language Kluwer Academic Publishers 2003.

[2] O'Neill, M., Brabazon, A., Adley C.: The Automatic Generation of Programs for Classification Problems with Grammatical Swarm, Proceedings of CEC 2004, Portland, Oregon (2004) 104 – 110

[3] Piaseczny, W., Suzuki. H., Sawai, H.: Chemical Genetic Programming – Evolution of Amino Acid Rewriting Rules Used for Genotype-Phenotype Translation, Proceedings of CEC 2004, Portland, Oregon (2004) 1639 - 1646.

[4] Ošmera, P., Šimoník, I, Roupec, J.: Multilevel distributed genetic algorithms. In Proceedings of the International Conference IEE/IEEE on Genetic Algorithms, Sheffield (1995) 505–510.

[5] Ošmera, P., Roupec, J.: Limited Lifetime Genetic Algorithms in Comparison with Sexual Reproduction Based GAs, Proceedings of MENDEL'2000, Brno, Czech Republic (2000) 118 – 126

[6] Li Z., Halang W. A., Chen G.: Integration of Fuzzy Logic and Chaos Theory; paragraph: Osmera P.: Evolution of Complexity, Springer, 2006 (ISBN: 3-540-26899-5) 527 – 578.

[7] Waldrop, M.M: Complexity – The Emerging Science at Edge of Order and Chaos, Viking 1993

[8] Purves, W. K. - Orians, G. H. - Heller, H. C.: Life, the science of biology, Sinauer Associates, Inc., 1992

[9] Ridley, M.: The Red Queen – sex and the evolution of nature, Penguin Books Ltd., 1993