Grammatical Evolution with Bidirectional Representation

Jiří Kubalík[1], Jan Koutník[2], and Léon J.M. Rothkrantz[3]

[1] Department of Cybernetics, CTU Prague, Technicka 2
166 27 Prague, Czech Republic
kubalik@labe.felk.cvut.cz

[2] Department of Computers, CTU Prague, Technicka 2
166 27 Prague, Czech Republic
koutnij@cs.felk.cvut.cz

[3] KBS Group, Department of Mediamatics, TU Delft
P.O. Box 356, 2600 AJ Delft, The Netherlands
L.J.M.Rothkrantz@its.tudelft.nl

Abstract. Grammatical evolution is an evolutionary algorithm designed to evolve programs in any language. Grammatical evolution operates on binary strings and the mapping of the genotype onto the phenotype (the tree representation of the programs) is provided through the grammar described in the form of production rules. The program trees are constructed in a pre-order fashion, which means that as the genome is traversed first the left most branch of the tree is completed then the second from the left one etc. Once two individuals are crossed over by means of simple one-point crossover the tail parts of the chromosomes (originally encoding the structures on the right side of the program tree) may map on different program structures within the new context. Here we present a *bidirectional representation* which helps to equalize the survival rate of both the program structures appearing on the left and right side of the program parse tree.

1 Introduction

Grammatical evolution (GE) is an evolutionary algorithm designed to evolve programs in any language, which can be described by a context free grammar [6], [9]. Unlike standard genetic programming (GP) [4], GE does not operate directly on the tree representation of the programs. It adopts an idea of phenotype-genotype encoding instead. The programs are represented as linear bit-string genomes and the correspondence between the phenotype and genotype is provided through the grammar described in the form of production rules.

The genotype is read piece by piece from left to right, generating integer values that are used to determine the production rules to be used. In standard GE the parse tree of the program is constructed in a depth-first manner, which means that as the genome is traversed first the left most branch of the phenotype is completed then the second from the left one etc. Once two individuals are crossed over by means of simple one-point crossover, the head parts of the chromosomes pass on the offspring with exactly the same meaning as they had in parents whilst the tail parts may map on different phenotype since their context changed. The consequence of this fact is that the "deeper" the program structures (partial solutions) are coded in the tail of the chromosome the less chance they will survive the crossover operation they have.

In this paper we present an enhancement of the standard GE that lies in so-called *bidirectional representation* which equalizes the survival rate of both the program structures appearing on the left and right side of the program parse tree. The paper starts with a brief description of GE. We focus on the aspect of the pre-order phenotype generation. Then the bidirectional representation is introduced and first experiments are presented. The paper concludes with discussion on the effect of the bidirectional representation.

2 Grammatical Evolution

GE is designed to evolve programs in any language, which can be described by a context free grammar. A commonly used notation for context free grammar is Backus Naur Form (BNF). BNF describes the grammar in the form of production rules that use terminals, which are the non-expandable items that are used in the language, i.e. $+$, $-$ etc. and non-terminals, which can be expanded into one or more terminals and non-terminals. A grammar can be represented by the tuple $\{N, T, P, S\}$, where N is set of non-terminals, T is set of terminals, P is a set of production rules that maps the elements of N to $N \cup T$, and S is a start symbol, which is a member of N. As an example let us take the BNF below with $\{N, T, P, S\}$ defined as follows

$$N = \{expr, op, pre-op, var\}$$

$$T = \{+, -, *, /, sin, cos, exp, log, X\}$$

$$S = expr$$

P :
<*expr*> ::= <*expr*> <*op*> <*expr*>
| <*pre-op*> <*expr*>
| <*var*>
<*op*> ::= +
| −
| *
| /
<*pre-op*> ::= *sin*
| *cos*
| *exp*
| *log*
<*var*> ::= *X*

The key to success and efficiency of GE lies in the method for generating and preservation of valid programs. GE does not work with a natural tree representation of the programs; instead it runs the evolution on binary strings. The genotype-phenotype mapping works so that the binary string is translated into a sequence of integers, called codons, each specifying the number of the production rule to be applied for currently expanded non-terminal. Thus the sequence of codons determines a sequence of choices to be made in the process of derivation of the program.

As the codon's value is typically larger than the number of available rules for the expanded non-terminal (codons are usually coded using 8 bits), the modulo of the codon's value to the number of rules is taken as the final rule choice:

$$choice = codon\ MOD\ number\_of\_rules$$

As a consequence of the use of the rule we can observe that multiple codon values can select the same rule. The role of such a redundancy in genetic code was studied in [5] and it turned out to be useful for proper functioning of GE.

Note it is possible for an individual to have more codons than needed for generating the complete program. In such a case the left over codons can be deleted. It is also possible that all codons have been read, but the generated tree still contains non-terminals to be expanded. If this happens the technique called wrapping can be used, which consists in the repeated reuse of the individual's codons. Obviously, using the described mapping mechanism only syntactically correct programs can be generated from any binary string.

The evolutionary algorithm of GE itself is quite simple. Standard genetic operators for binary representation are employed in GE. Namely the one-point crossover, called the *ripple crossover* in the context of GE [3], exhibits very good characteristics in terms of the generative and explorative capabilities. It can use a duplication operator, which randomly selects a number of codons that are placed at the end of the chromosome. For more detailed description of GE we refer the reader to [6].

The mapping process finishes as all of the nonterminal symbols have been replaced by terminals. In standard GE the left-most nonterminal out of the nonterminals to be processed is selected for substitution. Thus the program tree is constructed in a pre-order fashion, which means that as the genome is traversed first the left most branch of the phenotype is completed then the second from the left one etc. This becomes an important issue when looking closer at how new programs are generated by crossover operation.

3 Information Loss during Crossover in GE

Likewise other evolutionary algorithms, e.g. genetic algorithms (GA) and GP, GE searches for the optimal solution through the process of identifying some promising parts of the solution (partial solutions, in GA and GP called building blocks, [1], [2], [8]), which are grouped together in bigger and bigger parts finally assembled in the optimal solution. In order to work so the algorithm should be able to sample those good partial solutions with higher frequency than the useless ones when choosing the material for creating the candidate solutions. It should further be ensured that those important parts have high chance not to be disrupted through a recombination operation. This is called the exploitation ability of the algorithm. On the other hand the algorithm must be resistant against early stagnation of the population, which requires the algorithm to have enough exploration ability to effectively sample the solution space. Thus the best performance of any evolutionary algorithm is achieved when the optimal trade-off between its exploitation and exploration is found. The major impact on the generative or explorative power of the algorithm has the recombination operator. In standard GP crossover operators based on the single sub-tree swapping are used usually. It turns out that such operators cannot provide a sufficient information exchange in later stages of the run. Simply because the trees are already too large, usually containing a lot of meaningless code so just a single sub-tree exchange is not very likely to bring much new.

In GE a simple one-point crossover is used. An application of the operator on the linear chromosomes results generally in, what is called a rippling effect in [3], multiple sub-tree exchange between the involved parental individuals. Briefly said, the head part of the chromosome, i.e. that one before the crossover point, of each parent constitutes a spine of the program tree expression while the tail part represents number of sub-trees removed from the original tree. The offspring are then completed so that vacated sites of the spine of one mate are filled by sub-trees removed from the other mate and vice versa, see [3]. Moreover, the head parts of the chromosomes pass on the offspring with exactly the same meaning as they had in parents whilst the tail parts may map on different phenotype. This may happen if the sub-tree is grafted onto a site that is expecting a different symbol than that in the root of the sub-tree. The consequence of such a context dependent codon's interpretation, called *intrinsic polymorphism* of GE [3], is that the likelihood of transfer of a sub-tree from the parent to the offspring with a constant interpretation changes with its position within the original tree. The "left-most and shallow" structures (partial solutions) are less likely to be changed than the "right-most and deep" ones coded in the tail of the chromosome.

From this point of view, the crossover operation in GE should not be seen as the information exchange between two solutions as usually observed in standard GA and GP. It is rather partial re-initialisation of one solution with the code taken from the other one (that is likely to be re-interpreted at the new place).

The effect of one-point crossover is schematically depicted in Fig. 1. On average, a half of the tree is changed during the crossover. The offspring is given half of the code from the first and half of the code from the second parent. The black and white parts of the parental trees go unchanged to the offspring while the grey parts represent the probably wasted information, the structures which are likely to change their interpretation.

Apparently the amount of information damaged in each application of the crossover has a severe impact on the performance of the whole algorithm. As the evolution goes on only the left-most sub-tree structures (represented by the

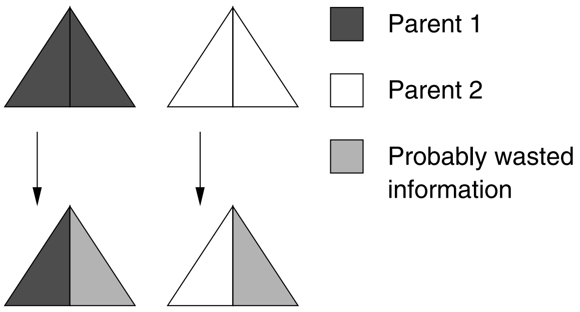

Fig. 1. One-point crossover with prefix encoding only

head-parts of chromosomes) are easily propagated to the successive populations. The sub-trees which are deep and in the right part of the tree have reduced chance to survive to the next population. This effect would become even more pronounced when the tree representation has strictly defined placement of the solution traits within the tree. Then the shallow and left structures would be formed, correctly identified and proliferated in the population prior to the deep and right ones. Thus the exploration of the whole search space would be biased by those "first-coming" partial solutions. Instead of a parallel exploration of the solution space the algorithm would seek the optimal solution in a partially sequential way - step by step. Clearly the risk of such an evolution become trapped in some sub-optimal solution increases.

In order to remedy this problem of preferring certain parts of the tree to the others and possible discarding the information during the crossover operation we introduce an extension of the standard GE representation as described in the following section.

4 Bidirectional Representation

In standard GE the programs are generated from a linear chromosome in a pre-order fashion as described above. So the linear encoding can be considered as a prefix expression of the program. In the extended GE so called bidirectional representation of the programs is employed. Each particular solution is encoded by two *paired chromosomes*: one expresses the program parse tree in a prefix notation and the other one in a postfix notation.

Now, when a pair of parental individuals is chosen to generate the offspring the crossover is applied twice (*i*) on the pair of their prefix chromosomes and (*ii*) on the pair of their postfix chromosomes. This results in four new chromosomes - two of them describing a couple of new programs in the prefix notation and two of them describing another couple of new programs in the postfix notation.

Finally the paired chromosome, representing the same program in the opposite way, must be constructed for each newly created chromosome (i.e. the prefix chromosome for the postfix one and vice versa). This is done in order to keep the

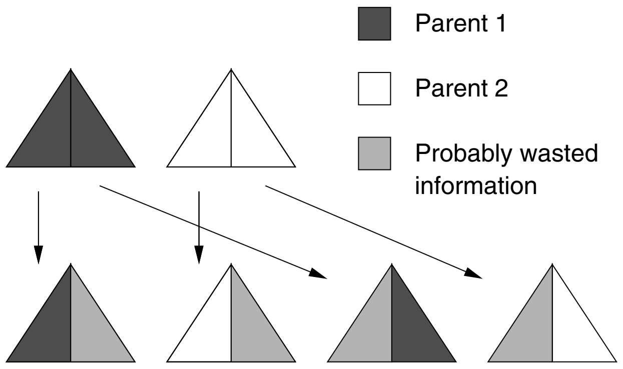

Fig. 2. One-point crossover with prefix and postfix encoding

prefix and postfix expressions for each individual consistent. However an issue arises as how to treat the newly generated under-specified individuals i.e. those with fewer codons in the chromosome than needed for complete generation of the program. In the current version a wrapping operator is invoked in such a way that the reused codons are appended at the end of the chromosome. Thus the paired chromosome describing the same tree must have the same number of codons.

The schema of the crossover performed on the enhanced representation is shown in Fig. 2. It illustrates the idea that using the bidirectional representation should better balance the exploitation of the information of the head and tail structures of the parental trees and so to reduce the losses of the information encoded in the tails of the "one-directional" chromosomes in original GE.

5 Setup of the Experiments

In order to test effect of the bidirectional representation in GE the concept was tested and compared to the standard GE. We have carried out series of experiments on the problem of symbolic regression as the test problem commonly used by the GP and GE community for evaluating the algorithms. We have used the following two target functions $X^4 + X^3 + X^2 + X$ and $X^5 - 2X^3 + X$. For both the fitness cases were taken from the interval [-1, 1]; 20 cases for the former function and 50 cases for the later one. Fifty runs were carried out with each algorithm on both problems. Both the standard GE and the bidirectional GE used the grammar from Sect. 2 and the configuration as follows:

- Raw fitness: the sum of the error taken over 20 respectively 50 fitness cases,
- Hits: the number of fitness cases for which the error is less than 0.01,
- $Populationsize$: 500,
- $Generations$: 200,
- $pmut$: 0.01,
- $pcross$: 1.0.

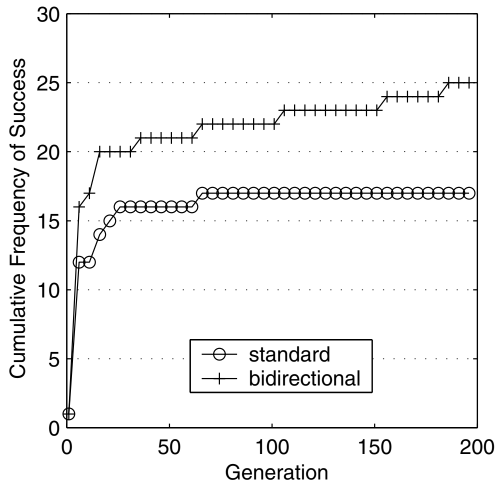

Fig. 3. Cumulative frequency of success obtained with standard GE and GE with bidirectional representation on $X^4 + X^3 + X^2 + X$

A commonly used condition under which different evolutionary algorithms are compared is the same number of fitness function evaluations. Note that in GE with bidirectional representation each crossover generates four new individuals instead of two as in standard GE. For the sake of simplicity a generational replacement strategy was used in the experiments where one generation means a replacement of a population by the completely new one. So the number of crossover operations needed for generating of the new population in bidirectional GE is of one half of the number of crossovers in standard GE while the number of performed evaluations is equal.

6 Results

First experiments clearly demonstrate the effect of bidirectional representation in GE. Figures 3 and 4 show a significant improvement in cumulative frequency of success obtained with bidirectional GE, which outperforms the standard GE in ratio 25/17 on the problem $X^4 + X^3 + X^2 + X$ and 12/4 on $X^5 - 2X^3 + X$. Steep beginnings of the plots in Fig. 3 may seem a bit surprising as there are quite some successes encountered already in $4^{th} - 5^{th}$ generation. This might be caused by rather robust initialization of the first population where the chromosomes up to 100-codon length are generated. Nevertheless, the difference in the explorative or generative power of the two approaches is distinct. Bidirectional GE exhibits a continuous progress in improving of the cumulative frequency of success whilst the standard GE does not yield anything new after $60^{th} - 70^{th}$ generation.

An interesting observation is that considering the formulas found by the GE. While on the first test problem the success formula almost always represented the target expression $X^4 + X^3 + X^2 + X$ on the later problem quite diverse

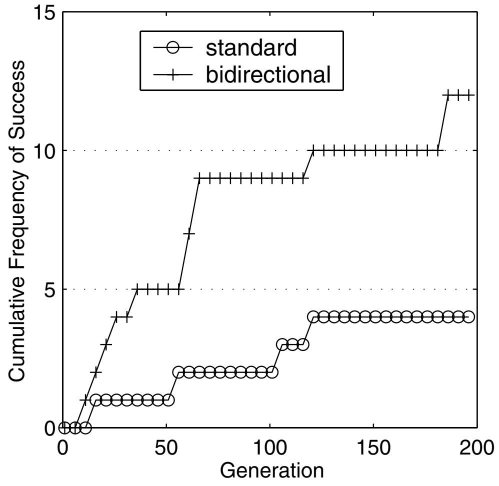

Fig. 4. Cumulative frequency of success obtained with standard GE and GE with bidirectional representation on $X^5 - 2X^3 + X$

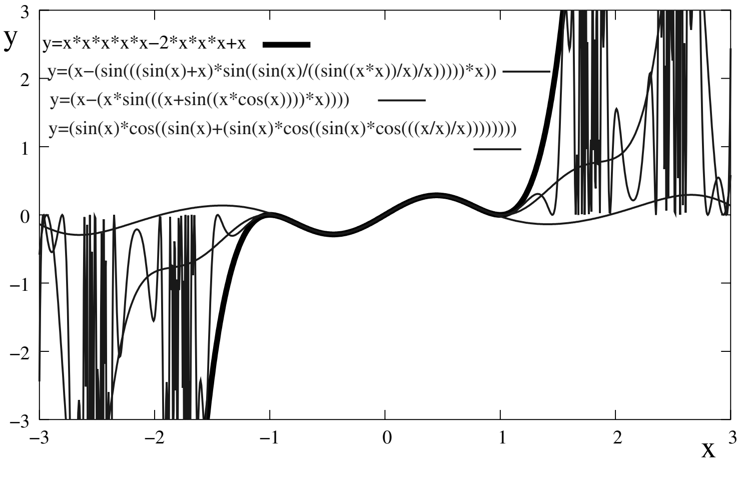

Fig. 5. Examples of functions that best approximate function $X^5 - 2X^3 + X$ within the interval [-1, 1] obtained with the GE with bidirectional representation

success formulas were produced. Some examples are given in Fig. 5. Note that all of them perfectly approximate the target function within the interval [-1, 1] while the error drastically grows up outside the interval.

The last figure shows cumulative numbers of improvements of the *best-so-far* observed between two successive populations. It illustrates the explorative power of bidirectional GE which exhibits sustained improvement of the *best-so-far* solution during the whole run.

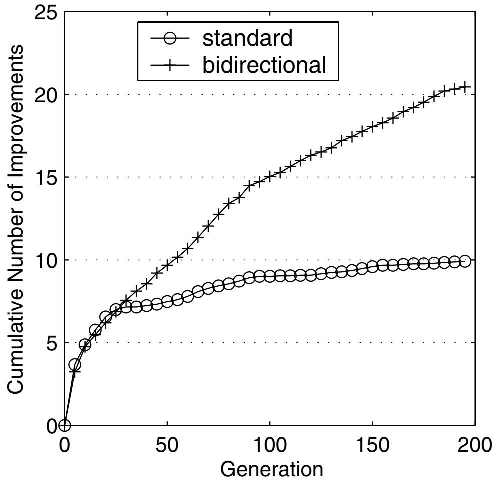

Fig. 6. Cumulative number of improvements of the *best-so-far* solution obtained with standard GE respectively GE with bidirectional representation on the problem $X^4 + X^3 + X^2 + X$. Only improvements observed between two successive populations are counted. Intra-generation improvements are not considered

7 Conclusions

In this paper we have introduced so-called *bidirectional representation* for GE. The representation has been proposed to help GE to exploit equally all the partial structures of the evolved programs.

The concept of the bidirectional representation has been tested on two instances of the symbolic regression problem. Results approved our expectations that the new representation should enhance the exploration and exploitation capabilities of GE. However, the conclusions are drawn from only two experiments in symbolic regressions. Other different types of problems should be tried in order to evaluate the concept to a satisfactory extent.

However some issues remain for further research such as how to treat the under-specified chromosome when building its equivalent paired chromosome. The version of the wrapping operator used in this work meet a requirement on consistent and equal paired chromosomes. On the other hand, this technique clearly causes the chromosomes to grow long leading likely to superfluous complex structures.

Acknowledgments

This research work is supported by the Grant Agency of the Czech Republic within the project No. 102/02/0132.

References

1. Angeline, P. J.: Subtree Crossover: Building Block Engine or Macromutation? Genetic Programming 1997: Proceedings of the Second Annual Conference, pp. 9-17, Morgan Kaufmann, 13-16 July 1997.
2. Goldberg D. E.: *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.
3. Keijzer M., Ryan C., O'Neill M., Cattolico M., and Babovic V.: Ripple Crossover in Genetic Programming. *Proceedings of EuroGP'2001*, LNCS, Vol. 2038, pp. 74-86, Springer-Verlag, 2001.
4. Koza, J.: *Genetic Programming.* MIT Press. 1992.
5. O'Neill, M., Ryan, C.: Genetic Code Degeneracy: Implications for Grammatical Evolution and Beyond, *Advances in Artificial Life*, LNAI, Vol. 1674, p. 149, Springer Verlag, 1999.
6. O'Neill, M., Ryan, C.: Grammatical Evolution. *IEEE Transactions on Evolutionary Computation*, Vol. 5 No.4, August 2001.
7. Poli, R., Langdon, W. B.: A Review of Theoretical and Experimental Results on Schemata in Genetic Programming, *Proceedings of the First European Workshop on Genetic Programming*, LNCS, Vol. 1391, pp. 1-15, Springer-Verlag, 14-15 April 1998.
8. Poli, R.: Hyperschema Theory for GP with One-Point Crossover Building Blocks, and Some New Results in GA Theory, Genetic Programming, *Proceedings of EuroGP'2000*, LNCS, Vol. 1802, pp. 163-180, Springer-Verlag, 15-16, 2000.
9. Ryan C., Collins J.J., O'Neill M.: Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Proceedings of the First European Workshop on Genetic Programming* , LNCS 1391. pp. 83-95. 1998.