Importance of selecting adequate grammars for specific problems within grammatical evolution

Christian Dobnik

School of Computer Science & Informatics
University College Dublin

Abstract

In grammatical evolution the selection of the grammar for a specific problem can naturally affect the solution at the very end. Optimizing the parameters for the evolutionary algorithm is – for grammatical evolution – only one part of getting good or even the best results. The selection of the grammar should be taken very seriously, the more complex a grammar is designed, the bigger is the search space for finding a solution. The experiments will show, that problems should be evaluated carefully at the beginning to find the right and simple grammar for the specific problem.

1. Introduction

Papers within grammatical evolution and other biologically inspired algorithms often describe optimization experiments for various parameters of the different algorithms. This paper is not different, but it focuses on a parameter, that is very often untreated, the grammar. A solution for a specific problem can derive from many different grammars, but only a few well-designed grammars can force the algorithm to find the right (or best) solution.

[Brabazon 2006] describes, that grammatical evolution extends the biological analogy by employing principles from genetics that have been uncovered by molecular biologists. Grammatical evolutions most important new feature in comparison to other biological algorithms is the mapping from genotype to phenotype as modeled from nature. In nature the genotype (the genetic material stored in the DNA) contains instructions to control the development of a living organism (the phenotype). These instructions get extracted from the genetic material and are different to the molecules responsible for the phenotype (proteins).

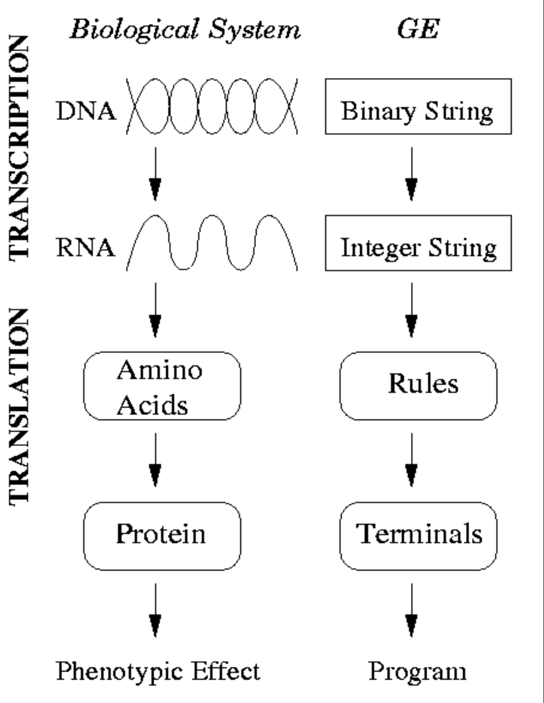

Fig. 1: Grammatical evolution and the biological system (from www.grammatical-evolution.org)

As stated in [O'Neill 2003] in grammatical evolution, the genotype is represented as a binary string provided by a search engine, a grammatical algorithm. This binary string corresponds to the DNA in the nature. This DNA is transcribed to RNA, in grammatical evolution the binary string is transformed to an integer string. Each integer corresponds to a 8-bit codeon from the binary string. Now the mapping process, using the grammar, starts: The integer string is mapped to rules from a provided grammar. In nature a similar process takes place: the RNA is transformed to a sequence of amino acids. These acids are the basic modules for proteins. In grammatical evolution the rules together represent a program or a function. This program or individual can then be executed and tested to see if it meets the requirements. In nature the proteins describe a specific effect on the living organism, imaginable the color of the eyes of a human being or the of fingers on the left hand. The whole process is also presented in figure 1 as a diagram from [GE 2006]. It also shows the two stages transcription, where the binary string is finally transcribed to an integer string, and translation, where the integer string is translated to a specific individual.

The search engine is in the normal case a genetic algorithm, so parameters as the probability of crossover and mutation, the population size and the number of generations has to be optimized within grammatical evolution to be sure to get the best results.
But not only genetic algorithms are used for the grammatical evolution, also alternatives like the particle swarm have been adopted to create the new approach grammatical swarm.

1.1 The grammar as an untreated parameter

As stated in [Wotawa 2006] grammars describe languages, more precisely grammars define possible words and sentences. Naturally not all possible sentences of a language will be written down, rules will be established, which allow to build sentences out of the words. So it is possible to construct a infinite number of sentences with a finite number of words.
In grammatical evolution a special type of grammar is used, the BNF or Backus-Naur form. As described in [Knuth 1964], BNF is a metasyntax used to express context-free grammars.

[Brabazon 2006] describes, that within grammatical evolution a suitable BNF grammar definition has to be defined. “The BNF can either be the specification of an entire language or, perhaps more usefully, a subset of a language geared the problem at hand.” Unfortunately, no description or explanation is given on that. As stated in the beginning, a solution can derive from many different grammars. The derivation tree and so the genotype can be very different to each other, but they can build the same sentence and therefor represent a right solution. The question is, how much the design of a grammar effects the search of a solution. Which mistakes should be avoided when designing a grammar for the grammatical evolution algorithm?
In Table 1 two grammars are presented. They both can construct the same sentences in the language, but the derivation tree is slightly different. Because the derivation tree is different, also the genotype will be different. It is hard to guess, which genotype would be found faster by an genetic algorithm, as in this particular case every bit can change the individual at the end.

<expr> := 01<expr> | 01	<expr> := 0 <expr1> | 01 <expr1> := 1 <expr> | 1

Table 1: Two different grammars with the same sentences in the language

Often, it is hard to find a simple grammar for a specific problem. Within grammar design and building compilers, there are several rules, how to get a simpler grammar out of a complex one. For further understanding of grammars and how to work with them, [Wotawa 2006] could be useful. Also compiler principles and techniques are useful to understand the importance of simple grammars.

1.3 Related Work

Researchers focus on optimizing the parameter for the genetic algorithm and show for specific problems, with which parameters they get the best result. In [O'Neill 2004] the idea of meta-grammars was adopted. A meta-grammar is a grammar that describes a construction of another grammar. “This grammatical evolution by grammatical evolution (GE^2) approach allows the grammar itself to be evolved and has been shown to be particularly effective in dynamic environments” [Brabazon 2006]. The mGGA (the meta-grammar genetic algorithm) has been described, that can automatically compose blocks of symbols that can be reused to construct a solution. In this paper, the focus is on grammars, that are designed by hand, no meta-grammar is used. If the grammar is well designed, it will probably also be better than self-constructed grammars by the algorithm.

2. Experiments

Within the two experiments described below, the libGE-library in version 0.26 for grammatical evolution [GE 2006] was used. For the search engine the suggested galib in version 2.4.6 was used. The experiments were designed to test following hypothesis:

Hypothesis: The simpler a grammar is designed, the faster and better the grammatical evolution algorithm finds the best solution.

2.1 Setting for the experiments

For all experiments, the same parameters for the genetic algorithm was used. The population size was 1000 individuals, the algorithm ran for only 10 generations. Probability of crossover was 90% and for mutation 1%. Five runs were performed. The parameters are also listed in Table 2. Nearly all parameters are default parameters of the genetic algorithm, because they were not very important. The main focus of the experiments is to show the difference of the various grammars.

Parameters for the GA	
population size	1000
number of generations	10
probability of crossover	0.9
probability of mutation	0.01

Table 2: Parameters for the experiments, if parameters are not listed, the default parameter was used.

2.1.1 The fitness function

The fitness function used in the experiments is a rather dummy function. It calculates the difference of input and output, sums up all 20 test-cases, inverts this score and adds the quantity of correct calculations. Shortly, the fitness score is most of the time between 0 and 1, only if it gets the correct solution, the score is 20 to indicate the best solution.

2.1.2 The different grammars

In every experiment, four different grammars were used to show the difference of the behavior of the grammatical evolution algorithm. In the experiments, symbolic regression problems were used to test different grammars. These symbolic regression problems chosen for the experiments only need addition and multiplication operations. The first grammar has a very simple design and fits the specification to be able to construct such basic symbolic regression problems.

```
<expr> ::= <expr> <op> <expr>
         | <var>
<op>   ::= + | *
<var>  ::= X
```

Listing 1: Simple Grammar

The start symbol for this recursive grammar is <expr>. Such an expression can either become a sequence of an expression an operator and another expression (<expr> <op> <epr>) or simply a variable (<var>). The non-terminal operator-symbol (<op>) can become the addition- or the multiplication-sign (+ or -). The variable-symbol can only become the variable “X”.
The disadvantage of the first grammar is the design of the first non-terminal <expr>. In almost 50 % of the time, it will become “var” initially, because the algorithm can choose between two possibilities. So the result “X” will appear very often as an individual. With exchanging the non-terminal <var> directly with “X”, the grammar would become even shorter, also the derivation tree will slightly become shorter, but for expandability and flexibility the grammar was designed like this.

The second grammar forces the sentence to grow to a minimum depth of 2. The start symbol "expr" can only be transformed to the sequence <var> <op> <expr1>, what means, that the solution "X" is no longer possible. It would be possible, if the start symbol would be <expr1> instead if <expr>, but the grammar was especially designed to avoid this sort of individuals. The <expr1>-symbol can recursively transform into <expr>, so that the sentence can also grow until infinity like the first one. The operators and variables haven't changed.

```
<expr>  ::= <var> <op> <expr1>
<expr1> ::= <expr>
         | <var>
<op>    ::= + | *
<var>   ::= X
```

Listing 2: Grammar 2, forces growth

The third grammar is not very different to the second one, it only shows its difference in the derivation tree, because its design is slightly more complicated as the second grammar. The only difference really is, that <expr1> can not be transformed into <expr> anymore. The size of the language is exactly the same as with the second grammar. Also, if the <expr> line would be deleted, the functionality of the rules would be the same and so the language. With deleting the first line, the grammar would also become very similar to the first grammar, what naturally was intended. All the grammars should be very similar and should nearly only vary in the derivation tree for the sentences.
It also forces the growth to depth two and gets the same result as the second grammar, but the derivation tree should be much smaller.

```
<expr>  ::= <var> <op> <expr1>
<expr1> ::= <var> <op> <expr1>
         | <var>
<op>    ::= + | *
<var>   ::= X
```

Listing 3: like Grammar 2 but with smaller derivation tree

The fourth grammar is also similar to the second one, but now some noise is introduced. A third operator sign, the subtraction, can be used, as a variable the constant number "1" is added to the variable-list.

```
<expr>  ::= <var> <op> <expr1>
<expr1> ::= <expr>
         | <var>
<op>    ::= + | * | -
<var>   ::= X | 1
```

Listing 4: Grammar with "noise"

The last grammar can cover the largest language of all, since it has two more terminals, an operator and a variable. Elsewhere it has no other differences to the grammars shown above. To find the right and best solution, the genetic algorithm has to search a lot more as with the other grammars, which present the real basic rules for constructing the symbolic regression problems.

2.1.3 Test cases

In every experiment every individual was tested with 20 test cases, simply the input-values from 1 to 20. Every individual was represented as a C++-file, was compiled by the grammatical evolution algorithm and tested.

2.2 Experiment 1

The first task for the grammatical evolution was to find the simple solution 4x² or as it would derive from the grammars: `X * X + X * X + X * X + X * X`. Here two things are very important for the grammatical evolution algorithm. First the correct genotype has to be able to grow to the correct depth and second the codeons for the operators must alternate, one bit determines whether it is a multiplication or an addition.

2.2.1 Results for Grammar 1

The maximum score was 1.9997, which means, that the best individual had nearly two right outputs for the input values. The computed best individual looked like: X * X + X + X + X * X * X, which was already found in generation three, but the genetic algorithm could not evolve better individuals after that. The average of all scores was at 0.0511.

2.2.2 Results for Grammar 2

The algorithm found the same best individual as with grammar 1, so the maximum score with 1.9997 was the same. The best individual was already found in the initial population. With 0.0971 the average of all scores was much higher than with grammar one. The force to start with a depth of two could be the reason for the better average, because the high percentage of individuals that are presented as an "X" will effect the overall average score a lot.

2.2.3 Results for Grammar 3

Also the third grammar didn't change the results. The algorithm found the same individual, the average of all scores was slightly unde
r that from grammar 2 with 0.0811. Also here the force to grow to depth 2 is clearly visible to have a positive effect on the solution.

2.2.4 Results for Grammar 4

With the noise of grammar four, the results were clearly different compared to the first three tests. The best individual (`X - X - 1 + X + 1 * X`) gets only a score of 1.00012, the average of all scores was only 0.02108.

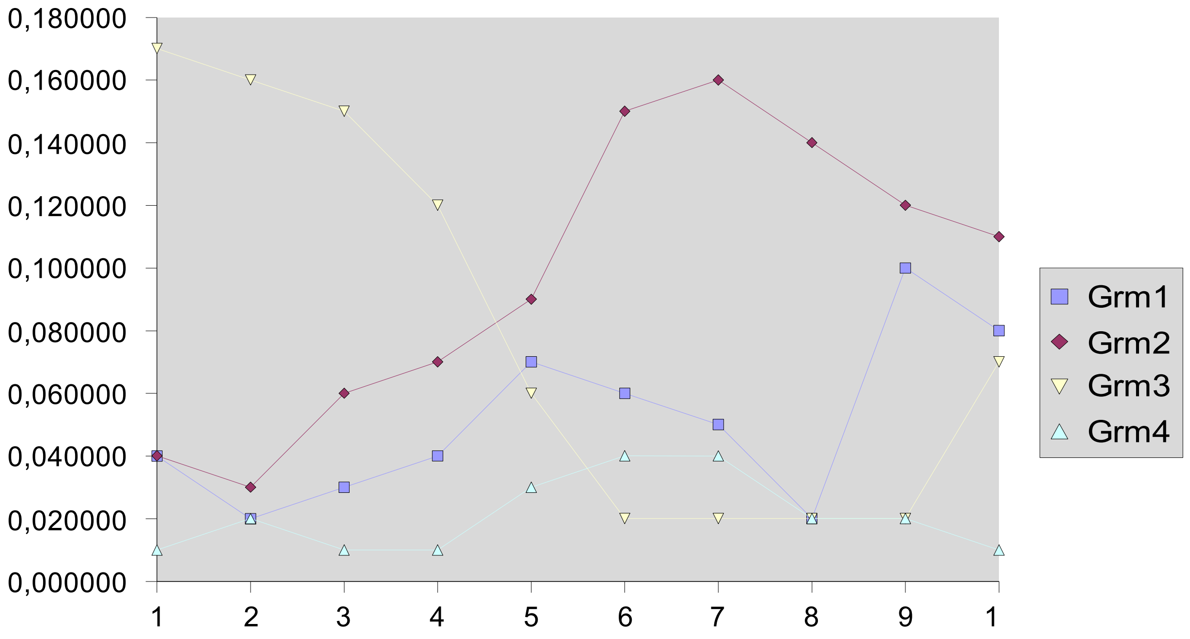

Fig. 2: performance of the mean scores for experiment 1

2.2.5 Conclusion for experiment 1

Experiment 1 did not show, that the simplest grammar is the best for this problem, because the algorithm found the same individual with the first three grammars. Figure 2 shows, that, from generation number 5, the average score was the best in the tests with grammar one.
One thing, that you can see from these results is, that noise in the grammar, say terminals, that you will not need in the solution, can have a very high effect on the found solution. In this experiment the test with grammar 4 was the worst in nearly every generation. So this shows that a complex grammar should not be chosen for such kind of problems, since it seems to complicate the search for a good individual since the search base is much grater.

2.3 Experiment 2

Since the results from experiment one are not very satisfying, a new experiment was started. Now, the problem is also simple but more complex to derive it: $x^4 + x^3 + x^2 + x$. Now the sentence has to grow even longer, the search should take longer to find a good individual. The settings were naturally the same as in experiment one. As in experiments one, all four grammars are able to construct the solution in demand.

2.3.1 Results for Grammar 1

Grammar one found the correct individual (`X * X * X * X + X * X + X * X * X + X`) in the first tun, so it found the maximum score of 20. The average score of the whole population over the five runs was only 0.057001 due to the fact, that also in this case many "X"-individuals were produced. The discovery of the right solution was only a "lucky punch". Like in experiment one, the mean score in the first nine generations increases slightly, with the discovery of the right solution it gets its highest point with 0.23.

2.3.2 Results for Grammar 2

Grammar two only had a best score of 1.00002, presenting `X * X + X + X + X * X * X * X` as its best individual. The average score was, like in experiment one, better than with grammar one. The average of all scores was at 0.07899. Also here, the mean score increases slightly through the generations. Although the algorithm did not find the best solution, the mean score in the last generation is with 0.24 a little bit higher than with the first grammar.

2.3.3 Results for Grammar 3

Also grammar three did not find the best solution. With the individual `X * X + X * X + X * X * X * X + X` it also only got 1.00002 score-points. With a average score of 0.0430014 it is also not as good as the second grammar, although the grammars seem to be the same. In this test, the mean score is going up and down through the generations with its highest score in generation 4 with 0.0700018 score points.

2.3.4 Results for Grammar 4

Like in experiment one, the test with grammar four did not perform very well. The maximum score was only 1 and the average of all scores was down to 0.0220014. This confirms, that noise in the grammar has a bad affect on the solution.

2.3.5 Conclusion for experiment 2

In experiment two, only the test with the first grammar found the right solution. As stated in the result, this discovery is probably the luck of the genetic algorithm, because the mean values were very low in each generation.

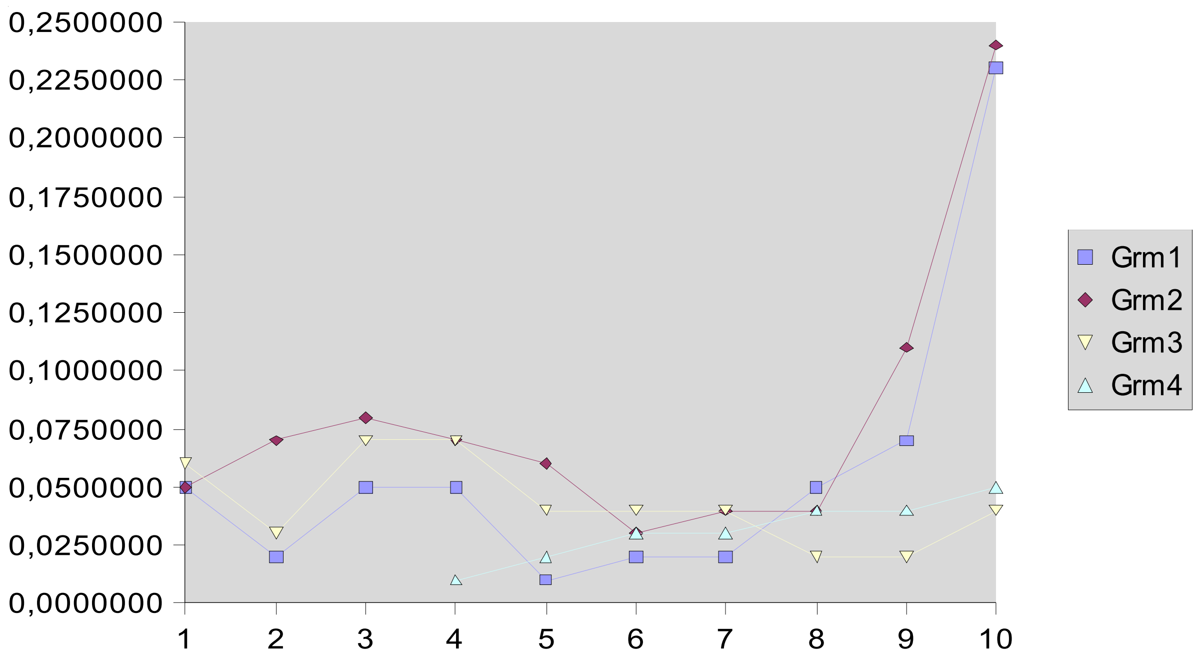

Fig. 3: performance of the mean scores for experiment 2

In figure 3 the mean scores for every grammar in every generation is plotted. Grammar 2, although it does not find the correct solution after 10 generations, performs best and has the best mean score over all ten generations.

3. Discussion

The two experiments were very similar and so provided nearly the same behavior. Both experiments

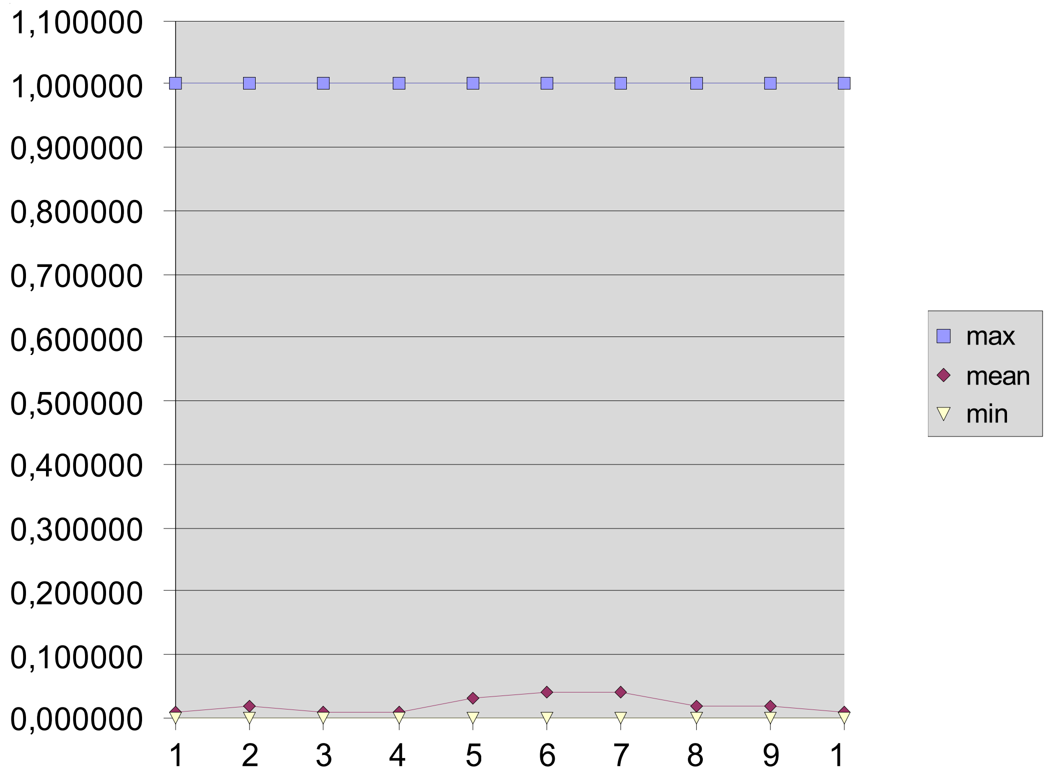

Fig. 4: max, mean and min score for grammar 4 in experiment 1

show, that a poorly-designed grammar with terminals not needed, like grammar 4, have an affect on the solution because the search space is much higher than with the other grammars. With a genetic algorithm, that would be optimized for such symbolic regression problems, the results would be much more better, the best solution would be found faster and more often. As this paper focus on the grammars itself, this optimization of the genetic algorithm was ignored and default parameters were used.

It is hard to guess, whether the algorithms would have performed better with optimization of the parameters, future work will have to investigate the behavior. I assume, that, like in the experiments here, the grammar with noise will always perform not as good as the simple one.
Grammar 4, with one more operand and a additional constant, had never the chance to perform better than the other grammars. Figure 4 shows, that the maximum score was always around 1, the minimum and mean score where very low, the mean score was always under 0.045. There is no visible performance or dynamic in the results of grammar 4, what shows, that such poorly-designed grammars have a tendency to perform as bad as they can.

The experiments performed quite similar with the three other grammars, because the grammars themselves were very similar. Out of these three, the second grammar, which forced the algorithm to grow to a depth of 2, performed best according to the mean score over all generations. The results of the third grammar differ a little bit to the results of the second one, which has probably to do with the different derivation of the sentences. Would the experiments go on with more generations and better parameters, I assume that grammar 2 and 3 would also find the best solution.

4. Conclusion

The two experiments overall not really satisfy the hypothesis, that the simplest grammar will always perform best. Although grammar one has found the right solution in experiment two, the mean score did not perform as good as with grammar 2. Otherwise, the results show, that poorly-designed grammars are not useful for finding good solutions, since the performance, as we see in figure 4, does not increase during the whole evolving of new generations.

So first, we can definitely summarize, that grammars should be well designed within grammatical evolution. Every terminal, that is not needed in the solution, should not be used in the rules of the grammar. So the hypothesis made in section 2 is half way right.

What about the statement, that always the simplest grammar will perform best? On the first view on the results, this is also true since grammar one is the only one, that could find the correct solution in experiment two. So we could say, that the hypothesis is true. If we have a deeper view on the results and compare the mean scores of the grammars, we can see, that the performance of grammar two is better and the solution of grammar one was probably only luck.

The reason, that grammar two performs better is obvious. Grammar two forces the algorithm to grow to depth two initially, so, the grammar shows the algorithm already the right way at the beginning. So grammar two is in fact the simplest grammar for the presented symbolic regression problems, because it contains all terminals needed without noise and starts already with depth 2, what, in fact, is a big advantage to grammar one, where 50 % of the individuals will be “X”.

So overall, we can say, that the two experiments showed, that the hypothesis in section two, that the grammatical evolution algorithm will perform better and will find the solution faster, if the grammar is simple and well designed, is true.

5. Future Work

For the future, there are several areas to think about and several experiments to perform to underline the correctness of the hypothesis. First of all, the experiments should run with optimized parameters for the genetic algorithm. This will show, that with optimized search engine, the grammar is still very important to find correct solutions.
Further, the fitness function could be updated to be able to show better visualizations of the performance of the experiments. Another attempt would be, to change the search engine and find out, if there would be a difference in the performance of the different grammars.

References

[Brabazon 2006] Brabazon A., O’Neill M. 2006. *Biologically Inspired Algorithms for Financial Modelling.* Springer.

[GE 2006] www.grammatical-evolution.org, visited 22.11.2006

[Knuth 1964] Knuth D. - 1964, *Backus Normal Form vs. Backus Naur Form* in Communications of the ACM 7

[O'Neill 2003] O'Neill, Ryan C. - 2003, *Grammatical Evolution. Evolutionary Automatic Programming in an Arbitrary Language.*

[O'Neill 2004] O’Neill M., Ryan C. - 2004. *Grammatical Evolution by Grammatical Evolution: The Evolution of Grammar and Genetic Code.* in Proceedings of EuroGP 2004.

[Wotawa 2006] Wotawa F. 2006. *Software-Paradigmen – Skriptum zur Lehveranstaltung (Spftwareparadigms – Lecture Notes)*