Exposing a Bias Toward Short-Length Numbers in Grammatical Evolution

Marco A. Montes de Oca

IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
mmontes@ulb.ac.be

Abstract. Many automatically-synthesized programs have, like their hand-made counterparts, numerical parameters that need to be set properly before they can show an acceptable performance. Hence, any approach to the automatic synthesis of programs needs the ability to tune numerical parameters efficiently.
Grammatical Evolution (GE) is a promising grammar-based genetic programming technique that synthesizes numbers by concatenating digits. In this paper, we show that a naive application of this approach can lead to a serious number length bias that in turn affects efficiency. The root of the problem is the way the context-free grammar used by GE is defined. A simple, yet effective, solution to this problem is proposed.

1 Introduction

Genetic Programming (GP) [1] has been used for the automatic synthesis of computer programs and other kinds of systems. In many cases, a GP system is required to find two equally important components: a system's structure and optimal (or near-optimal) values for numerical parameters. Although both components determine the overall system's performance, it is its structure that determines the number and importance of numerical parameters [2]. Obviously, tuning numerical variables effectively and efficiently is crucial in any GP approach and the topic has been the subject of active research [3], [4], [5].

Previous work on the ability of Grammatical Evolution (GE) [6], a grammar-based GP technique, to synthesize numerical values has shown that a simple digit concatenation approach is superior to the traditional expression-based one [7]. In this paper, a study on the efficiency with which this approach is able to generate numerical parameters is presented. The study relies on the assumption that a good structure (i.e., the number of numerical variables) has already been found during evolution, so that the efficiency with which GE can tune numerical variables can be studied in detail.

The main finding reported here is that the classical digit concatenation grammar used by GE to generate numerical parameters induces a bias toward short-length numbers. This bias can affect substantially the efficiency of the search process which can hinder the applicability of GE as a whole. A simple, yet effective, solution to this problem is proposed. It consists of a grammar modification

that makes the distribution of lengths in the initial population more uniform, effectively making the search for numbers of different lengths more efficient.

The paper is organized as follows. Section 2 describes the GE approach. A brief summary of related work is presented in Section 3. Section 4 presents the number length bias problem. Section 5 describes the experimental setup used to evaluate both the magnitude of the problem and the benefits obtained with the proposed solution. The paper is concluded in Section 6.

2 Grammatical Evolution

Grammatical Evolution (GE) [6] is a recent evolutionary computation technique for the automatic synthesis of programs in an arbitrary language. At the core of the approach is a grammar-based mapping process that transforms a number of variable-length integer vectors into syntactically correct programs. The elements of an integer vector are used to select a production rule from a grammar defined in a Backus-Naur form. By expanding production rules in this way, a complete program can be generated.

The components of a solution vector are normally integers in the range $[0, 255]$. Their values are used to select a production rule[1] from the nonterminal symbol that is being expanded. The selected production rule is determined by

$$\text{selected rule} = (\text{integer value}) \, mod \, (\text{No. of rules for current nonterminal}) \,, \quad (1)$$

where *mod* denotes the modulus operator.

As an example of the operation of GE, consider the problem of performing symbolic regression. A grammar $G = \{N, T, S, P\}$ for this problem is shown below (taken from [8]). N is the set of nonterminal symbols, T is the set of terminal symbols, S is the start symbol, and P is the set of production rules.

$$\begin{aligned}
N &= \{< expr >, < op >, < func >, < var >\} \\
T &= \{\sin, \cos, \exp, \log, +, -, /, *, x, 1.0, (,)\} \\
S &= < expr > \\
P &= \{ \\
< expr > &\rightarrow < expr > < op > < expr > \\
&\quad | \, (< expr >) \\
&\quad | \, < func > (< expr >) \\
&\quad | \, < var > \\
< op > &\rightarrow + \mid - \mid / \mid * \\
< func > &\rightarrow \sin \mid \cos \mid \exp \mid \log \\
< var > &\rightarrow 1.0 \mid x \\
&\quad \}
\end{aligned}$$

[1] Rules are numbered starting from 0.

Suppose that the solution vector we want to map is

$$[6, 25, 120, 58, 43, 62, 126, 87, 67, 23, 11, 2]\,.$$

From the start symbol $< expr >$, there are 4 rules to choose from. Since the first element of the solution vector is 6, the selected rule is rule number $6 \, mod \, 4 = 2$. After this first expansion, the solution takes the form $< func > (< expr >)$. The mapping process continues by selecting a production rule from the leftmost nonterminal symbol, which in our example is $< func >$. In the next expansion step, the selected rule is rule number $25 \, mod \, 4 = 1$, so the solution takes the form $\cos(< expr >)$. If we continue with this process the final solution would be $\cos(\log(\exp(x))/1.0)$.

The mapping process is repeated until a string with no nonterminal symbols is generated or until no more elements in the vector remain to be mapped. If after processing all the elements of the solution vector a valid solution is still incomplete, there are two possible actions to take. The first one is called *wrapping* and consists in reinterpreting the solution vector again starting from the first element until a valid solution is generated or a maximum number of wrappings occur. Although the elements of the solution vector are reused, their effect on the generated string depends on the nonterminal symbol that is being rewritten. The second option is to discard the solution and assigning it the lowest fitness value.

By the way GE is designed, it is possible to separate the search and solution spaces. This has the advantage of decoupling the way search is done from the way solutions are constructed. Consequently, GE does not necessarily rely on genetic algorithms to work.

GE has been used in fields such as financing [9], combinatorial optimization [10] and machine learning [11]. In these and other cases a common problem stands out: synthesizing numerical values effectively and efficiently. Previous work on this direction is presented below.

3 Constant Creation by Grammatical Evolution

There have been some previous studies on the ability of GE to synthesize numbers. O'Neill et al. [7] presented a comparison between the traditional expression-based approach used in tree-based Genetic Programming with a digit concatenation one. Based on experimental evidence, they conclude that the digit concatenation approach is superior to the expression-based one on the problem of synthesizing numbers. In Dempsey et al. [12], a comparison between the digit concatenation approach and another one using random constants was performed on problems similar to those used by O'Neill et al. [7]. No conclusive evidence was found on the superiority of any of these approaches. Recently, a more detailed study was undertaken by Dempsey et al. [13] in which they finally conclude that the digit concatenation approach is superior to the random constants approach on problems requiring the synthesis of static constants. The random constants

approach proved to be better suited for dynamic problems (in which the target number changes over time).

Dempsey et al. [14] explored a meta-grammar approach to constant creation. The grammars that were used to create the potential solutions were evolved along with the solutions themselves. In this work, the effects of using different grammars were indirectly studied but no grammar analysis was conducted in detail and therefore, no grammar-construction guidelines were derived. Dempsey and colleagues found that the meta-grammar approach offered some advantages over other approaches on dynamic problems.

In this paper, we focus on the efficiency of the digit concatenation approach. It uses a grammar that includes the basic building blocks for number construction. For example, a grammar for synthesizing unsigned integer numbers is presented below.

Digit Concatenation Grammar

$$
\begin{aligned}
N &= \{< number >, < digitlist >, < digit >\} \\
T &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
S &= < number > \\
P &= \{ \\
< number > &\rightarrow < digitlist > \\
< digitlist > &\rightarrow < digit > \mid < digit >< digitlist > \\
< digit > &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
&\quad \}
\end{aligned}
$$

This same approach, with small changes in the grammar, can be used for creating signed and floating-point numbers. Note that, in principle, it is possible to build numbers of any length. However, we will see later that this grammar induces a bias toward short-length numbers, making the pure digit concatenation approach inefficient when high-precision numbers are needed.

4 Number Length Bias in Digit Concatenation Grammars

The works described in Section 3 studied the ability of GE to find constants, the length of which (or of their components in the case of floating-point numbers) was short (between one and five digits) (see e.g., [13] and [7]). Interestingly, in their results one can notice that the error after several generations is still quite high in the case of "long" constants (those with at least 5-digit-long components). Since the focus of these works was on the relative performance obtained by GE when using different approaches for constant creation, this phenomenon remained largely unexplained.

Large errors when trying to build long numbers can be explained using simple concepts from the theory of stochastic context-free grammars [15], in which each

production rule $r \in P$ has an associated probability $p(r)$ of being selected. Consider a normal application of GE that uses a context-free grammar (CFG). We can consider the probability of selecting a production rule whose left-hand side nonterminal symbol is X, to be

$$p(r) = \frac{1}{|X|}\,, \tag{2}$$

where $|X|$ is the number of production rules that have X as their left-hand side symbol. Since we are using CFGs, the probability of a complete derivation is simply the product of the rule probabilities used and thus, the probability of generating a particular string (in our case, a number) is the sum of the probabilities of all possible derivations producing that particular string. For example, the probability of generating the number 5261 (in a GE style) from the digit concatenation grammar presented before is

$$\begin{aligned}
&< number > \rightarrow\ < digitlist > && p_1 = 1.0\\
&< digitlist > \rightarrow\ < digit >< digitlist > && p_2 = 0.5\\
&\quad < digit > \rightarrow 5 && p_3 = 0.1\\
&< digitlist > \rightarrow\ < digit >< digitlist > && p_4 = 0.5\\
&\quad < digit > \rightarrow 2 && p_5 = 0.1\\
&< digitlist > \rightarrow\ < digit >< digitlist > && p_6 = 0.5\\
&\quad < digit > \rightarrow 6 && p_7 = 0.1\\
&< digitlist > \rightarrow\ < digit > && p_8 = 0.5\\
&\quad < digit > \rightarrow 1 && p_9 = 0.1
\end{aligned}$$

$$p(< number > \Rightarrow\ 5261) = p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 = 6.25 \times 10^{-6}\,.$$

The probabilistic view presented above applies only when we generate strings at random, which is the case at initialization. The search algorithm behind GE will then try to adjust the population so as to increase the individuals' fitness (or reduce error). However, from a practical point of view, initial fitness is very important because it determines to a great extent the efficiency (i.e., the speed of convergence) of the approach.

In general, the probability of generating in the initial population a number of n digits (without instantiating any digit to a specific number) using the digit concatenation grammar is $1/2^n$. Clearly, the longer the length of the target number, the less likely it is to have a good approximation of it in the initial population.

5 Experiments

In our experiments, we measure the relative error $\delta_{\hat{x}}$ of the best solution $\hat{x}$ with respect to the target number value x. It is computed as follows

$$\delta_{\hat{x}} = \frac{|\hat{x} - x|}{|x|}, \tag{3}$$

where $x \neq 0$, which is always the case in our experiments.

The relative error measure is used as the fitness evaluation which is to be minimized. It guides a steady-state genetic algorithm which is used as search algorithm. At each trial, a different target number of length n is randomly generated in the range $[10^{n-1}, 10^n - 1]$. The parameters of the algorithm and the experiments are listed in Table 1.

Table 1. Parameters settings used in our experiments

Parameter	Value
Search algorithm	Steady state GA
Crossover operator	One point
Mutation operator	Bit-wise mutation
Population size	100
Number of generations	100
Probability of crossover	0.9
Probability of mutation	0.01
Population replacement strategy	100%
Wrapping events	No
Number of trials	100

The steady-state genetic algorithm with a 100% population replacement strategy effectively behaves as an algorithm with a $(\mu + \lambda)$ replacement strategy where the best individuals among parents and offspring are passed over to the next generation.

Figure 1 shows the relative error after 1000 fitness evaluations obtained by GE as a function of the length of the target number when using the classical digit concatenation grammar presented in Section 3.

The relative error grows with the target number length reaching a maximum value of 1.0. A maximum value of the relative error equal to 1.0 means that the best solutions found after 1000 fitness evaluations are insignificant (in terms of value) with respect to the long-length target numbers. It should be noted that the effectiveness of the approach is not under discussion. After 10000 fitness evaluations, the algorithm was capable of finding the target number irrespective of its length; however, it should be stressed that the focus of this study is on efficiency. In this respect, the results show clearly that building numerical values by using the classical digit concatenation grammar induces a strong bias toward short-length numbers.

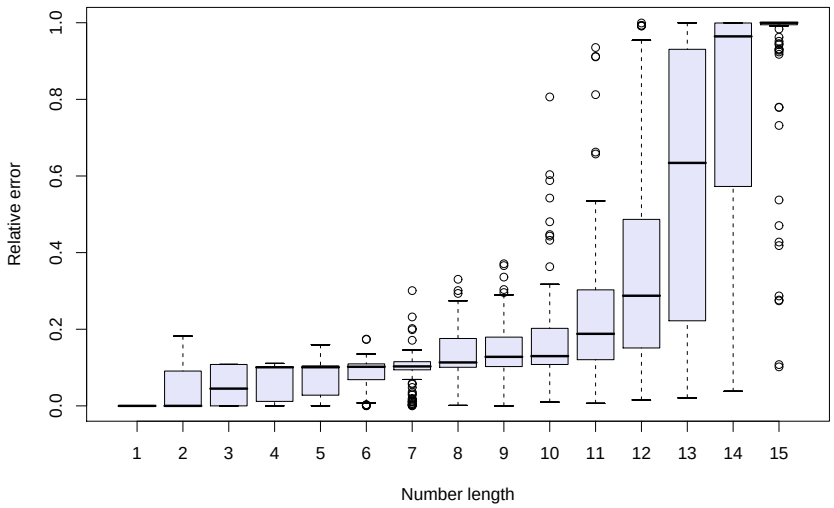

Fig. 1. Relative error as a function of the length of the target number. These results were obtained after 1000 fitness evaluations with the classical digit concatenation grammar.

The probability of having long-length numbers in the initial population decreases exponentially with the numbers' length. Ideally, this problem is solved by substituting the digit concatenation production rule by a rule with exactly the same number of digits as the target number. The rule in question is the following:

$$< number > \rightarrow \underbrace{< digit >< digit > \ldots < digit >}_{n \text{ digits}} .$$

By using this rule, the problem is reduced to select the appropriate value for each of the digits of the target number. Effectively, short- and long-length numbers (up to a length of n digits) are generated with equal probability in this way. Figure 2 shows the relative error obtained by GE as a function of the length of the target number using this exact-length grammar. Relative errors are small irrespective of the target number's length.

The solution just described suffers from one main drawback: It is necessary to estimate accurately the length of the target number before running the search process. If the length of the target number is greater than the estimation, the solution will fail miserably because it will not be possible to generate a number of the appropriate length. If the length of the target number is shorter than the

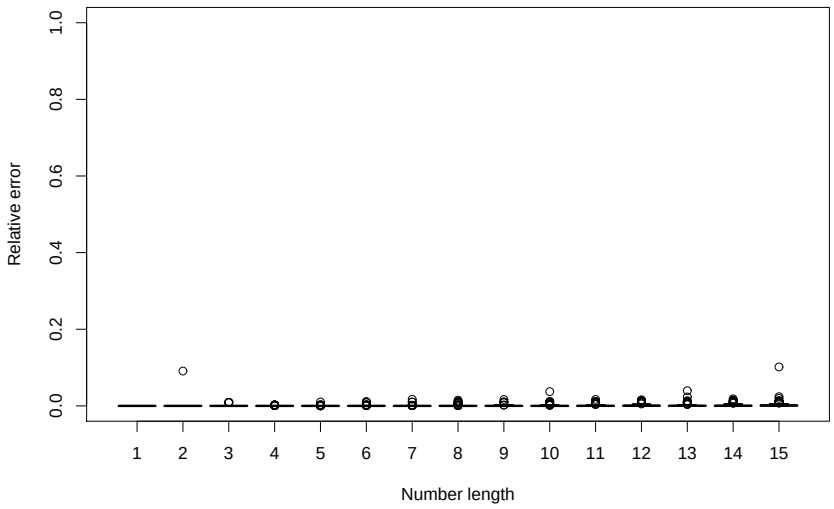

Fig. 2. Relative error as a function of the length of the target number. These results were obtained after 1000 fitness evaluations with an exact-length grammar. The grammar corresponds exactly with the length of the target number.

estimate, the relative error will grow exponentially with the overestimation as shown in Figure 3.

An intermediate solution is thus proposed. It reduces considerably the bias toward short-length numbers and eliminates the need of estimating the target number's length beforehand. The production rules involved in the digit concatenation process are the following:

$$
\begin{aligned}
&< number > \rightarrow < digitlist > \\
&< digitlist > \rightarrow < digit > \mid < digit >< digitlist > \\
&\qquad\qquad \mid < digit >< digit > \mid < digit >< digit >< digitlist > \\
&\qquad\qquad \ldots \\
&\qquad\qquad \mid \underbrace{< digit >< digit > \ldots < digit >}_{k \text{ digits}} \\
&\qquad\qquad \mid \underbrace{< digit >< digit > \ldots < digit >}_{k \text{ digits}} < digitlist >,
\end{aligned}
$$

where $k <= n$ and n is the target number's length. Figure 4 shows the relative error obtained with a "hybrid" grammar in which $k = 5$.

In general, the obtained error is much higher than the one obtained with an exact-length grammar, but lower than the one obtained with a pure digit

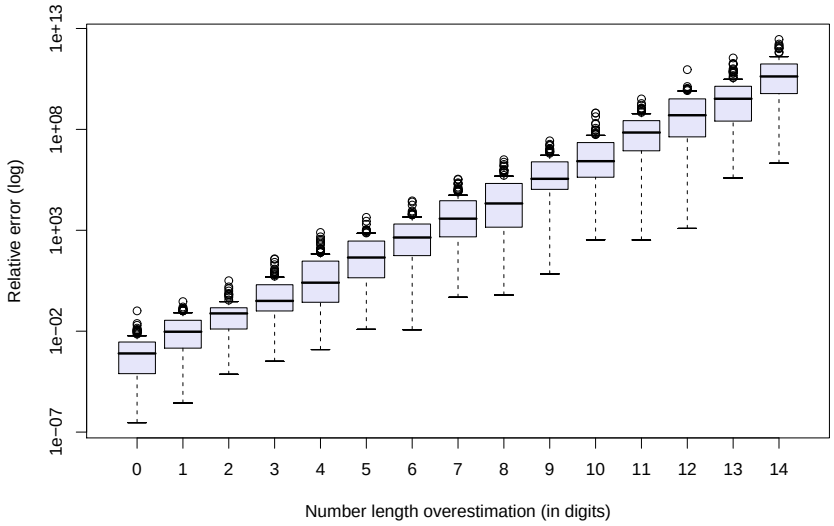

Fig. 3. Relative error obtained after 1000 fitness evaluations with an overestimated (in this case aimed at 15 digits) exact-length grammar. Note the logarithmic scale in the error axis.

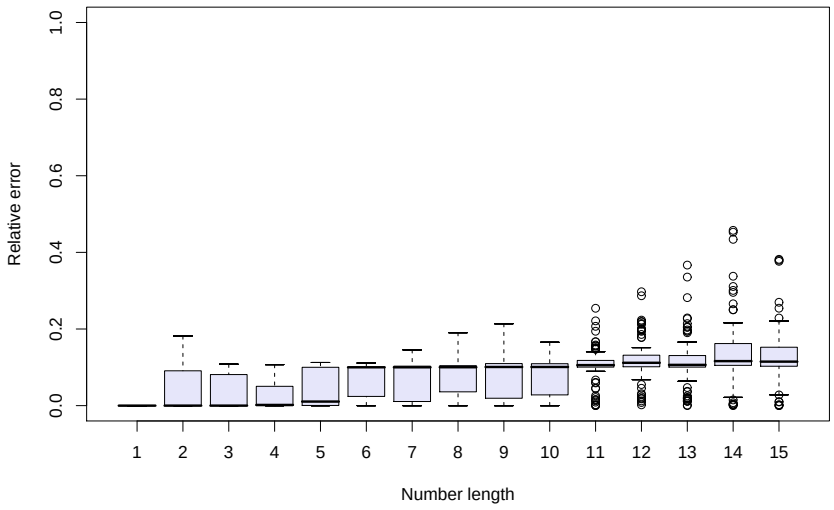

Fig. 4. Relative error as a function of the length of the target number obtained after 1000 fitness evaluations with a hybrid grammar.

concatenation grammar. The parameter k determines the maximum size of the building blocks available to GE to produce numbers. It is expected that the greater k, the more uniform the distribution of numbers of different lengths in the initial population becomes.

6 Conclusions

Grammatical evolution (GE) is a relatively new evolutionary algorithm for the synthesis of programs or systems in any arbitrary representation language. This is possible thanks to a grammar-based search.

This paper highlights the importance of properly defining the grammar used by GE for the solution of a problem. One of the main strengths of GE is the possibility of biasing the search by means of a grammar; however, undesired biases can also be introduced. Although the focus here was on the synthesis of numerical values, the analysis based on stochastic context-free grammars can be applied to determine whether there is any undesired grammar-induced bias on other applications. A careful grammar design is needed in all cases.

This and previous studies have focused only on the synthesis of one numerical value at a time. This has been done because a detailed understanding of the inner workings of grammatical evolution is necessary before embarking into more complicated studies. An investigation into the performance of GE on practically relevant application scenarios, in which several (not just one) numerical parameters are to be found, should be done in the future.

Acknowledgments

The author is grateful to Thomas Stützle, Mauro Birattari and the anonymous reviewers for their comments and suggestions to improve this paper. The author is funded by the Programme Alβan, the European Union Programme of High Level Scholarships for Latin America, scholarship No. E05D054889MX, and the *SWARMANOID* project funded by the Future and Emerging Technologies programme (IST-FET) of the European Commission (grant IST-022888).

References

1. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press, Cambridge, MA, USA (1992)
2. Koza, J.R.: Automatic synthesis of topologies and numerical parameters. In: Handbook of Metaheuristics. Fred Glover and Gary A. Kochenberger (editors). Kluwer Academic Publishers, Boston, MA, USA (2003) 83–104
3. Evett, M., Fernandez, T.: Numeric mutation improves the discovery of numeric constants in genetic programming. In Koza, J.R., et al., eds.: Genetic Programming 1998: Proceedings of the Third Annual Conference, Morgan Kaufmann (1998) 66–71

4. Topchy, A., Punch, W.F.: Faster genetic programming based on local gradient search of numeric leaf values. In Spector, L., et al., eds.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), San Francisco, CA, USA, Morgan Kaufmann (2001) 155–162
5. Li, X., Zhou, C., Nelson, P.C., Tirpak, T.M.: Investigation of constant creation techniques in the context of gene expression programming. In Keijzer, M., ed.: LNCS 3103. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004), Springer (2004) Late Breaking Paper.
6. O'Neill, M., Ryan, C.: Grammatical Evolution. Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers (2003)
7. O'Neill, M., Dempsey, I., Brabazon, A., Ryan, C.: Analysis of a digit concatenation approach to constant creation. In Ryan, C., et al., eds.: LNCS 2610. Proceedings of EuroGP 2003 – Sixth International European Conference on Genetic Programming, Berlin, Germany, Springer (2003) 173–182
8. O'Neill, M., Ryan, C.: Grammatical evolution. IEEE Transactions on Evolutionary Computation **5**(4) (2001) 349–358
9. Brabazon, A., O'Neill, M.: Biologically Inspired Algorithms for Financial Modelling. Springer, Berlin, Germany (2006)
10. Cleary, R., O'Neill, M.: An attribute grammar decoder for the 01 multiconstrained knapsack problem. In: LNCS 3448. Evolutionary Computation in Combinatorial Optimization, Berlin, Germany, Springer-Verlag (2005) 34–45
11. Tsoulos, I.G., Gavrilis, D., Glavas, E.: Neural network construction using grammatical evolution. In: Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology, Piscataway, NJ, USA, IEEE Press (2005) 827 – 831
12. Dempsey, I., O'Neill, M., Brabazon, A.: Grammatical constant creation. In Deb, K., et al., eds.: LNCS 3103. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004), Berlin, Germany, Springer-Verlag (2004) 447–458
13. Dempsey, I., O'Neill, M., Brabazon, A.: Constant creation in grammatical evolution. International Journal of Innovative Computing and Applications **1**(1) (2007) 23–38
14. Dempsey, I., O'Neill, M., Brabazon, A.: meta-Grammar Constant Creation with Grammatical Evolution by Grammatical Evolution. In: GECCO 2005: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, New York, NY, USA, ACM Press (2005) 1665–1671
15. Manning, C., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, MA, USA (1999)