Annales UMCS Informatica AI 7 (2007) 15-25

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Improved genetic algorithm for the context-free grammatical inference

Adrianna Gietka[*]

Institute of Computer Science, University of Gdańsk

Abstract

Inductive learning of formal languages, often called grammatical inference, is an active area in machine learning and computational learning theory. By learning a language we understand finding the grammar of the language when some positive (words from language) and negative examples (words that are not in language) are given. Learning mechanisms use the natural language learning model: people master a language, used by their environment, by the analysis of positive and negative examples. The problem of inferring context-free languages (CFG) has both theoretical and practical motivations. Practical applications include pattern recognition (for example finding DTD or XML schemas for XML documents) and speech recognition (the ability to infer context-free grammars for natural languages would enable speech recognition to modify its internal grammar on the fly). There were several attempts to find effective learning methods for context-free languages (for example [1,2,3,4,5]). In particular, Y.Sakakibara [3] introduced an interesting method of finding a context-free grammar in the Chomsky normal form with a minimal set of nonterminals. He used the tabular representation similar to the parse table used in the CYK algorithm, simultaneously with genetic algorithms. In this paper we present several adjustments to the algorithm suggested by Sakakibara. The adjustments are concerned mainly with the genetic algorithms used and are as follows:

- we introduce a method of creating the initial population which makes use of characteristic features of context-free grammars,
- new genetic operations are used (mutation with a path added, 'die process', 'war/disease process'),
- different definition of the fitness function,
- an effective compression of the structure of an individual in the population is suggested.

These changes allow to speed up the process of grammar generation and, what is more, they allow to infer richer grammars than considered in [3].

1. Introduction

The aim of this work was to design an effective algorithm to generate a context-free grammar of an unknown language L_{CF} provided two sets of words

[*]*E-mail address*: adag@mig.gda.pl

are given: a set of positive examples S_+ (words from the language L_{CF}) and a set of negative examples S_- (words that are not in L_{CF}). The output of the procedure which uses genetic algorithms is a context-free grammar of L_{CF}, with minimal sets of nonterminals and productions. In section 2 we introduce some basic definitions and the idea of Sakakibara [3] on tabular representation. Section 3 contains the description of two versions of the implemented algorithm. Section 4 presents the results of the computer experiments carried out. The last section contains a brief summary and some conclusions.

2. Preliminaries

Definition 1. *A context-free grammar* (CFG) is a quadruple $G=(N,\Sigma,P,S)$, where N is an alphabet of nonterminal symbols, Σ is an alphabet of terminal symbols such that $N\cap\Sigma=\emptyset$, P is a finite set of production rules of the form $A\to\alpha$ for $A\in\Sigma$ and $\alpha\in(N\cup\Sigma)^*$, and S is a special nonterminal called the start symbol. For $\beta,\gamma\in(N\cup\Sigma)^*$ we write $\beta\Rightarrow\gamma$ if there exists a production $A\to\alpha$ and $\beta = \gamma_1 A\gamma_2$, $\gamma = \gamma_1\alpha\gamma_2$, for some $\gamma_1,\gamma_2\in(N\cup\Sigma)^*$. By $\Rightarrow^*$ we denote the reflexive and transitive closure of $\Rightarrow$. The language generated by CFG denoted by $L(G)=\{w\in\Sigma^*: S\Rightarrow^* w\}$ is called *a context-free language* [6].

Example 1. Let $G=(N,\Sigma,P,S)$, where $N=\{S,X\}$, $\Sigma=\{a,b,c\}$, $P=\{S \to Sc \mid aXbc, X \to aXb \mid \lambda\}$, then G generates a context-free language $L_1=\{a^nb^nc^m : n,m\geq 1\}$.

Definition 2. We say that a context-free grammar $G=(N,\Sigma,P,S)$ is in *Chomsky normal form* if all the productions are of the form $A\to B$ or $A\to a$ where $A,B,C\in N$ and $a\in\Sigma$.

Example 2. For the language L_1 the context-free grammar in Chomsky normal form can be defined as follows $G_{CF}=(N,\Sigma,P,S)$ where $N=\{S,A,B,C,X,Y\}$, $\Sigma=\{a,b,c\}$ and $P=\{S \to SC \mid XC, X \to YB \mid AB, Y \to AX, A \to a, B \to b, C \to c\}$.

For each context-free language L_{CF} such that $\lambda\notin L_{CF}$ there exists a grammar G in Chomsky normal form such that $L(G) = L_{CF}$ (λ stands for the empty word).

Definition 3. A representative sample of G, is defined to be a finite subset of $L(G)$ that exercises every production rule in G, that is, every production is used at least once to generate the subset. We denote the representative sample by R_+.

Example 3. For the language $L_1=\{a^nb^nc^m : n,m\geq 1\}$ and the grammar G_{CF} from example 2 we have the following representative samples:

1) $R_+=\{aabbcc\}$
   $S \Rightarrow SC \Rightarrow (XC)C \Rightarrow (YB)CC \Rightarrow (AX)BCC \Rightarrow A(AB)BCC \Rightarrow^* aabbcc$
2) $R_+=\{aabbc,abcc\}$
   $S \Rightarrow XS \Rightarrow (YB)C \Rightarrow (AX)BC \Rightarrow A(AB)BC \Rightarrow^* aabbc$
   $S \Rightarrow SC \Rightarrow (XC)C \Rightarrow (AB)CC \Rightarrow^* abcc$

Definition 4. *The Cocke-Younger-Kasami parsing algorithm* (CYK algorithm) is a deterministic algorithm for verifying whether some word $w=a_1a_2\dots a_n\in\Sigma^*$ can be derived in the context-free grammar $G=(N,\Sigma,P,S)$ in Chomsky normal form. In a finite number of steps we produce a triangular table of size $n\times n$ consisting of some subsets of N. A nonterminal A appears in the ith column and the jth row if $A\Rightarrow^* a_ia_{i+1}\dots a_{i+j-1}$, then it derives a subword of length j starting with a_i. Finally, the start symbol S appears in 1st column and nth row if and only if $w\in L(G)$.

Example 4. For the grammar G_{CF} (from example 2) and the words $w_1=abcc\in L_1$ and $v_1=aabb\notin L_1$ we have the following tables:

4	***S***			
3	*S*			
2	*X*			
1	*A*	*B*	*C*	*C*
	a	*b*	*c*	*c*

4	*X*			
3	*Y*			
2		*X*		
1	*A*	*A*	*B*	*B*
	a	*a*	*b*	*b*

Let us observe that if the grammar is unknown but we assume that each word has a derivation in some context-free grammar in a Chomsky normal form, then we can create a tabular representation inserting different nonterminals into each place in the table.

Definition 5. *The tabular representation* for the word w in the grammar, denoted by $T(w)$, is a triangular table $[t_{i,j}]$ filled with subsets of nonterminals in the following way:

- $t_{i,j}$ for $j=2,\dots,n$, $i=1,\dots,n+1-j$ consists of j-1 different nonterminals, that is $\{X_{i,j,k_1}, X_{i,j,k_2},\dots, X_{i,j,k_{j-1}}\}$,
- $t_{i,1}$ for $i=1,\dots,n$ consists of a singleton $\{X_{i,1,1}\}$.

The tabular representation $T(w)$ defines the grammar $G_T=(N,\Sigma,P,S)$ such that

$$N=\left\{X_{i,j,k}\,\middle|\,2\le j\le n,\ 1\le i\le n-j+1,\ 1\le k<j\right\}\cup\left\{X_{i,1,1}\,\middle|\,1\le i\le n\right\}\cup\{S\}$$

$$P=\left\{X_{i,j,k}\to X_{i,k,l}X_{i+k,j-k,m}\,\middle|\,2\le j\le n,\ 1\le i\le n-j+1,\ 1\le k<j,\ 1\le l<k,\ 1\le m\le j-k\right\}$$

$$\cup\left\{X_{i,1,1}\to a_i\,\middle|\,1\le i\le n\right\}\cup\left\{S\to X_{1,n,k}\,\middle|\,1\le k<n\right\}$$

Example 5. The tabular representation for $w=a_1a_2a_3a_4$, where $a_i\in\Sigma$ for $i=1,\dots,4$ is as follows:

$X_{1,4,1}$, $X_{1,4,2}$, $X_{1,4,3}$			
$X_{1,3,1}$, $X_{1,3,2}$	$X_{2,3,1}$, $X_{2,3,2}$		
$X_{1,2,1}$	$X_{2,2,1}$	$X_{3,2,1}$	
$X_{1,1,1}$	$X_{2,1,1}$	$X_{3,1,1}$	$X_{4,1,1}$
a_1	a_2	a_3	a_4

For example, the following productions are included in this representation:
$X_{1,4,1}\to X_{1,1,1}X_{2,3,1}\mid X_{1,1,1}X_{2,3,2}$

$X_{2,3,2} \rightarrow X_{2,2,1}X_{4,1,1}$
$X_{4,1,1} \rightarrow a_4$

Variables which are not used are denoted by 0 in our representation. For example for words $w_1=abcc \in L_1$ and $w_2=aabbc \in L_1$ we have the following representation, using the production rules from grammar G_{CF}, example 2:

S,0,0			
S,0	0,0		
X	0	0	
A	B	C	C
a	b	c	c

S,0,0,0				
X,0,0	0,0,0			
0,0	0,Y	0,0		
0	X	0	0	
A	A	B	B	C
a	a	b	b	c

3. Applying genetic algorithms to learn CFG

We assume that a finite set S_+ of positive examples (which contains a representative sample of some grammar for the unknown language L_{CF}) and a finite set S_- of negative examples, are given. We try to find a grammar G_{CF} in Chomsky normal form which generates the language L_{CF} and contains the minimal number of nonterminals and productions. To achieve our goal we employ a genetic algorithm (GA) which is a search technique in the space of alternative solutions of the problem [7].

Two versions of the algorithm were implemented and used in experiments. They differ in the structure of individuals of the population and also in the applied genetic operations.

3.1. Learning algorithm using full tabular representation

Here we list the main steps of the algorithm (the details are further in the paper):

1. Select a minimal pseudo-representative sample R_+ from the set of positive examples S_+.
2. Find the tabular representation for words from R_+, create the initial population of individuals and calculate the fitness of each individual.
3. While the fitness of the best individual in the population does not satisfy the termination criteria repeat the following steps:
   3.1. Using the roulette wheel method create the ‘new’ population of individuals from the current population.
   3.2. For individuals from the ‘new’ population perform:
      i. crossover on pairs of individuals,
      ii. mutation of individuals,
      iii. mutation with erasing,
      iv. mutation with a path added.

3.3. Calculate the fitness function for 'new' population.
3.4. Merge 'new' and 'current' population:
  i. increase the age of individuals in the population and remove some oldest individuals,
  ii. if the size of the population exceeds the predefined limit perform "war/disease process" (see [8] for details).

Selecting a minimal pseudo-representative sample

In order to select a minimal pseudo-representative sample $R_+ \subset S_+$ we proceed as follows. We select words from S_+, one by one, starting from the shortest one and proceeding to the longer ones. For each word, treated as a one element representative set, we generate a random initial population. If, in a population generated for a word, there exist individuals with non-zero fitness function which represent grammars generating some other words from S_+, then those words can be excluded from R_+. The aim of this method is getting the possibly smallest representative sample for our language, but our result may not be the optimal solution for our language, and for that reason we notice the achieved set pseudo-representative.

The structure of individuals

An individual is a vector of bytes corresponding to identifiers of variables from the tabular representation of all the words from the representative sample plus one byte to remember the age of an individual. With no loss of generality we can assume that productions of type $A \to a$ are fixed and for each terminal $a \in \Sigma$ there is one-to-one correspondence with a nonterminal $A \in N$. In this way we can delete variables $X_{k,1,1}$ (from the first row in the tabular representation) from the individuals. Similarly, variables in the last row $X_{1,n,k}$ where n is the length of the word and $1 \le k < n$, finally would be identified with the start symbol S and for that reason they are counted as one nonterminal when calculating the fitness function.

The size of an individual is

$$1 + \sum_{j=1}^{|R_+|} S\left(w_j\right),$$

where

$$w_j \in \mathrm{R}_+$$

and

$$S\left(w\right) = \sum_{i=1}^{|w|-1} i \cdot \left(|w| - i\right).$$

Example 6. Assume $R_+=\{w_a,w_b\}$, where $w_a=a_1a_2a_3a_4$, $w_b=b_1b_2b_3$ and $a_i,b_j\in\Sigma$. Then the structure of individual is:
$(Age, Xa_{1,2,1}, Xa_{2,2,1}, Xa_{1,3,1}, Xa_{1,3,2}, Xa_{3,2,1}, Xa_{2,3,1}, Xa_{2,3,2}, Xa_{1,4,1}, Xa_{1,4,2}, Xa_{1,4,3}, Xb_{1,2,1}, Xb_{2,2,1}, Xb_{1,3,1}, Xb_{1,3,2})$
where $Xi_{j,k,l}$ corresponds to a respective variable in the tabular representation for the word w_i. Besides, the special value 0 corresponds to the variables eliminated in the process of learning.

This structure is very similar to that used in [3]. The difference lies in an additional gene for storing the 'age' of individual (used by the 'die process' function) and also omitting the fixed variables described above.

Fitness function

For an individual p which represents the grammar $G_p = (N_p, \Sigma, P_p, S)$ we define

$$f(p)=\begin{cases} 0, & \text{if there exists } w\in S_- \text{ accepted by } G_p \\ \dfrac{c_1\cdot f_1(p)+c_2\cdot f_2(p)+c_3\cdot f_3(p)+c_4\cdot f_4(p)}{c_1+c_2+c_3+c_4}, & \text{otherwise}\end{cases}$$

$$f_1(p)=\frac{\left|\left\{w\in S_+ \middle| w\in G_p\right\}\right|}{|S_+|},$$

$$f_2(p)=\frac{1}{|N_p|},$$

$$f_3(p)=\frac{1}{|P_p|},$$

$$f_4(p)=\frac{1}{|NP_p|},$$

where NP_p is the set of non-zero variables from p, c_1, c_2, c_3, c_4 – the fixed constants chosen according to the importance of functions $f_1,\ldots,f_4$.

The introduction of components 3 and 4 (additional ones, as compared to [3]) allow for a better selection of grammars.

Generating the initial population

In the case of random generation of individuals, as used by Sakakibara, there exists a risk of lack of individuals with non-zero fitness function. In our algorithm we use a different approach. For each word in the representative sample one derivation path is randomly chosen. The nonterminals from this derivation appear in the individuals, other variables are equal to 0. This procedure is repeated *PopulationSize* times. To speed up the process of learning we expand the process of derivation path generation by adding some probability productions which lead to loops characteristic of stack operations in the pushdown automata:

– $X \rightarrow ZY$, $Y \rightarrow XV$ (and similarly, $X \rightarrow YZ$, $Y \rightarrow VX$),
– $X \rightarrow ZY$, $Y \rightarrow VX$ (and similarly, $X \rightarrow YZ$, $Y \rightarrow XV$).

where $X,Y,Z,V \in N$, $x,w,y,v,z \in \Sigma^*$ and $n \in IN$.

Both types of rules are characteristic of the context free languages with the same number of fixed subwords.

Genetic operators

– Crossover of two individuals means either an exchange or merge of two derivations of a word from R_+. In the case of exchange, we select randomly a word, and appropriate parts of individuals are exchanged. In the case of merging two derivations, all null variables in random derivation in one individual are replaced by corresponding non-null variables in the second one,
– Mutation of an individual – all occurrences of two randomly chosen nonterminals are replaced by a randomly chosen one of them,
– Mutation with erasing – means deleting one or all occurrences of a randomly chosen nonterminal,
– Mutation with a new path added – for a randomly chosen word from R_+ one new derivation is added (similarly to adding path in generating a new population process).

'Die process'

The population ages until the maximal age *MaxW* is achieved. For each individual the chance to survive is proportional to $(MaxW - Age)/MaxW$. When a new individual is created then its age is defined as 0 in order to survive at least one generation. The best individual in the population survives no matter what age it is.

'War/disease process'

As follows from the above the population size can vary. In each generation the individuals with the fitness value equal 0 are eliminated from the population. It is similar, for individuals exceeding the maximal age. In the case when the size of the population is bigger than *PopulationSize* the war/disease process is performed which means eliminating the weakest individuals.

3.2. Learning algorithm using a modified individual structure

The algorithm described above proved successful in testing several benchmark context-free languages. New challenge was set by the international contest Omphalos Competition in 2004 [9] (http://www.irisa.fr/Omphalos/), where the context-free inference problem was defined for more complicated and complex languages with a bigger set of nonterminals and productions.

Since the smallest language of the contest was described by 255 positive words and 535 negative words over a 5-symbol alphabet $\{b,c,d,e,f\}$ with the maximal length of 64 symbols for words in S_+, our algorithm of section 3.1 proved to be excessively time and space consuming. The following modifications of the algorithm were introduced.

A modified structure of an individual

We assume that the length of a word from S_+ does not exceed 255 and there are not more than 255 nonterminals in the grammar (both assumptions seem to be reasonable). Moreover, because of the larger size of S_+, we omit the initial process of selecting a minimal pseudo-representative sample, which is in this case more time-consuming and less effective. We start with the assignment $R_+=S_+$. Then for each word $w \in R_+$ we will store one derivation path in an individual using 2*(|w|-1)-1 bytes in the following way. The first |w|-1 bytes are used to store the structure of a selected path and in the remaining |w|-2 bytes we remember all the respective nonterminals.

For the word *aabbcc* generated by the grammar G_{CF} from example 2 we have:

5	4	3	1	1	*S*	*X*	*Y*	*X*

which corresponds to the following tabular representation:

	1	2	3	4	5	6
6	- - - - *S* [**5**]					
5	- - - ***S*** [**4**]	- - - -				
4	- - ***X*** [**3**]	- - -	- - -			
3	***Y*** [**1**] -	- -	- -	- -		
2	-	***X*** [**1**]	-	-	-	
1	*A*	*A*	*B*	*B*	*C*	*C*
	a	*a*	*b*	*b*	*c*	*c*

An individual in the population, as before, contains the representation of all words from R_+.

To compare the sizes of individuals in two versions of the algorithm see the table below. In the table we compare the number of bytes needed to store a derivation path for one word *w* of the length $|w|$ in the first and second algorithms.

$\lvert w\rvert$	2	3	4	6	8	10	15	20	25	30	40	56
First algorithm	1	4	10	35	84	165	560	1330	2600	4495	10660	29260
Second algorithm	1	3	5	9	13	17	27	37	47	57	77	109

Generating the initial population

Since bigger representative samples describe richer grammars with longer production sets the previous method of generating individuals might lead to many grammars with the zero fitness value. Such individuals are useless in the genetic algorithm. Therefore the procedure of generating an individual is as follows:

```
1. find a random derivation path for a word from $R_+$ and calculate the fitness of
   the individual
   1.1. if the fitness is non-zero then continue step 1 for the next word from $R_+$
   1.2. otherwise, exchange the nonterminals in the added derivation for those
        which were not used in individual yet and calculate the fitness
        1.2.1. if the fitness is non-zero then continue step 1 for the next word
               from $R_+$
        1.2.2. otherwise, we omit the derivation of this word in the individual and
               continue step 1 for the next word from $R_+$
```

Genetic operations

– Crossover of two individuals is performed similarly to version 1, thus for a randomly chosen word from R_+ appropriate parts of derivations are exchanged.
– Mutation and mutation with erasing affect only the part of an individual containing the nonterminals in order to sustain the structure of the derivation path.

4. Some results of computer experiments

4.1. Results for the algorithm from section 3.1

The experiments were performed for several context-free languages treated as typical or difficult by the community (benchmark problems from [2,3]). We assumed that terminal productions are fixed. For example for the alphabet $\{a,b,c\}$ the productions $A\rightarrow a$, $B\rightarrow b$, $C\rightarrow c$ are fixed. Some of the results are listed bellow.

1) Context free language $L_1=\{a^n b^n c^m : n,m \geq 1\}$
$S_+=\{w \in L_1 : |w| \leq 8\}$, $S_-=\{w \in \Sigma^* : w \notin L_1 \wedge |w| \leq 10\}$

	Pseudo-representative sample	Generation	Grammar
1	$\{abcc, aabbc, aabbcc\}$	15	$S \rightarrow SC \mid XC,\ X \rightarrow YB \mid AB,\ Y \rightarrow AX$
2	$\{aabbcc\}$	0^*	$S \rightarrow XC,\ C \rightarrow CC,\ X \rightarrow YB \mid AB,\ Y \rightarrow AX$
3	$\{abc, aabbc, aabbcc\}$	5	$S \rightarrow XC,\ C \rightarrow CC,\ X \rightarrow AY \mid AB,\ Y \rightarrow XB$

* grammar generated as an individual in the initial population

Since for each grammar more than one experiment was performed below we present the final grammar for the appropriate generation step.
2) Regular language $L_2=\{ac^m : m \geq 1\} \cup \{bc^m : m \geq 1\}$
Generation: 0
Grammar: $S \rightarrow SC \mid AC \mid BC$
3) Context-free language $L_3=\{w \in \Sigma^* : \#_a w = \#_b w\}$, $\Sigma=\{a,b\}$
Generation: 1, 7
Grammar: $S \rightarrow AB \mid BA$, $A \rightarrow AS$, $B \rightarrow BS$
4) Context-free language $L_4=\{ww^R : w \in \Sigma^*\}$, $\Sigma=\{a,b\}$
Generation: 6, 30
Grammar: $S \rightarrow XB \mid AA \mid BB \mid YA$, $X \rightarrow BS$, $Y \rightarrow AS$
5) Context-free language $L_5 \subset \{(,)\}^*$ – the language of proper parentheses
Terminal productions: $A \rightarrow ($, $B \rightarrow)$
Generation: 2, 4, 11
Grammar: $S \rightarrow AB$, $A \rightarrow AS \mid SA$
6) Context-free grammar $L_6=\{w \in \Sigma^* : \#_a w = 2\#_b w\}$, $\Sigma=\{a,b\}$
Generation: 2, 30, 87
Grammar: $S \rightarrow BX \mid XB \mid YA$, $X \rightarrow AA$, $Y \rightarrow AB$, $B \rightarrow SB$, $A \rightarrow SA$

4.2. Results for the algorithm from section 3.2

We will describe the results obtained for test 1 from the Omphalos Competition considered as a benchmark provider.

1. The first step aimed at generating a grammar deriving all 255 positive examples using fewer than 255 nonterminals (estimated maximum size of nonterminal set is 10524). In the 761 generation we found a grammar which accepted all positive examples, rejected all negative examples and used 102 nonterminals and 349 productions. What might be interesting is that the grammar found in the 194 generation accepted 254 positive words and it

needed about 500 further generations to correct the grammar so that all 255 positive words were accepted.

2. The objective of the second step was to minimize the number of nonterminals and productions. In the 3916 generation the grammar with 35 nonterminals and 114 productions was found. Since there was no progress in the next 1460 generations the experiment was terminated. The obtained grammar was tested on 518 words offered by the Omphalos Competition webpage and the results were not correct for all of them. Unfortunately, the webpage does not offer the final grammar to compare with our result. Nevertheless, the grammar found by our algorithm classifies correctly all positive and negative examples, and is strictly context-free (it includes self-embedding productions [10], so it isn't a regular grammar).

5. Summary and conclusions

The results obtained so far seem encouraging. The modifications introduced for the second algorithm that is economizing on the length of the individual, the method of generating of the initial population and the introduction of an improved genetic algorithm allowed to experiment with rich and difficult data sets. There is still space for the improvement, specially the careful selection of a minimal representative sample which was not performed in the second version of the algorithm and might prove useful. These conclusions will be the motivation for future work.

References

[1] Boden M., Wiles J., *On learning context-free and context sensitive languages*. IEEE Tran. Neural Networks, 13(2) (2002).
[2] Nakamura K., Matsumoto M., *Incremental learning of context free grammars based on bottom-up parsing and search*. Pattern Recognition, 38 (2005) 1384.
[3] Sakakibara Y., *Learning context-free grammars using tabular representations*. Pattern Recognition, 38 (2005) 1372.
[4] Clark A., *Learning deterministic context free grammars: The Omphalos Competition*. Mach Learn, 66 (2007) 93.
[5] Higuera C., *A bibliographical study of grammatical inference*. Pattern Recognition, 38 (2005) 1332.
[6] Hopcroft J.E., Ullman J.D., *Introduction to automata theory, languages, and computation*. Publisher Addison-Wesley, (2001).
[7] Goldberg D.E., Genetic Algorithms in Search, Optimization, and Machine Learning, Publisher Addison-Wesley (1989)
[8] Shi X., Liang Y., Lee H., Lu C., Wang L., *An improved GA and a novel PSO-GA-based hybrid algorithm*. Information Processing Letters, 93 (2005) 255.
[9] Starkie B., Coste F., van Zaanen M., *The Omphalos context-free grammar learning competition*. LNAI, 3264 (2004) 16.
[10] Nederhof M.J., *Practical experiments with regular approximation of context-free languages*. Association for Computational Linguistics, 26(1) (2000).