Iterative method of generating artificial context-free grammars

Olgierd Unold
Department of Computer Engineering
Wrocław University of Science and Technology
27 Wybrzeże Wyspiańskiego,
50-370 Wrocław, Poland
olgierd.unold@pwr.edu.pl

Agnieszka Kaczmarek
Wrocław Univ Sci & Technol
agnieszka.kaczmarek@pwr.edu.pl

Łukasz Culer
Wrocław Univ Sci & Technol
lukasz.culer@pwr.edu.pl

Extended abstract of the 14th International Conference on Grammatical Inference *ICGI'18*, September 5–7, 2018, Wrocław, Poland.

Abstract

There exist numerous grammar inference methods, that use sets of both positive and negative examples as an algorithm input. The origin of these examples could be very diverse - from real-life data to manually crafted data. Both categories have their advantages and disadvantages. We present an alternative approach: the application of an automated grammar generator.

Author Keywords

Grammatical Inference; context-free grammars; grammar generator.

Real-life data, as learning sets, promises the greatest performance in industrial applications if grammars are inferred properly. However, due to an imperfection of measurement equipment, some examples could include errors. Moreover, some of the phenomenon, that express through those data cannot be covered with formal language theory methods.

On the opposite side are sets for manually crafted grammars. Despite many advantages, like possessing full knowledge about them or certainty that examples are error-free, they also create some issues - creating a grammar of given complexity with positive and negative learning sets is difficult and time-consuming task.

$$\begin{cases} \frac{R_P^-}{S_T^2} \leq S_{NT} \\ \frac{R_P^-}{S_T^2} \leq R_P^+ + R_I + 2R_B + 1 \\ \sqrt{\frac{R_P^+}{S_T^2}} \leq S_{NT} \\ \sqrt{\frac{R_I}{2S_T}} \leq S_{NT} \\ \sqrt[3]{R_B} \leq S_{NT} \end{cases} \tag{1}$$

where

$$\begin{cases} S_{NT}, S_T, R_P^- \in \mathbb{N}_+ \\ R_P^+, R_I, R_B \in \mathbb{N}_0 \end{cases}$$

In this paper we would like to present our, currently in final stage of development, iterative method of generating artificial context-free grammars, that allows creating a consistent context-free grammar of given parameters automatically, with positive and negative example sets. The algorithm consists of several steps. Firstly, a grammar of given parameters is generated using our original approach. Then it used to generate proper positive and negative sets. [2].

The core part of the algorithm is the grammar generator. It takes as input given parameters: the exact number of parenthesis rules with non-terminal symbol (R_P^+),the exact number of parenthesis rules without non-terminal symbol (R_P^-), the exact number of branch rules (R_B), exact number of iterative rules (R_I), the maximum number of terminal symbols (S_T) and the maximum number of non-terminal symbols (S_{NT}). Alternatively, all those parameters could be replaced with a Grammar Complexity Index (GCI). It is intended to be a simple indicator of grammar complexity and is defined as the sum of all grammar rules ($|R|$) and was created to allow the generation of many grammars similar in complexity, conserving their structural diversity. The rule types were selected based on paper by Sakakibara The procedure starts with adding all parenthesis rules without a non-terminal symbol. During creation, the algorithm randomly chooses whether to create a new symbol (preserving the parameter requirements). Then, all other rules are created randomly using existing symbols or creating new ones. The creation of a non-terminal symbol is allowed only if it will be attached to the left-hand side of the new rule and the previously added non-terminal symbol would be applied to the right-hand side at least once. This approach ensures that all symbols are productive. The procedure has also to verify, that all a left-hand side symbols of parenthesis rules without non-terminal symbols are connected. The last step is a conversion of the recently added non-terminal symbol into a start symbol. This conversion makes all productive symbols achievable, which results in a consistent grammar.

A mathematical analysis of this procedure provided full insight into its properties. One of them was the possibility to create a grammar using given parameters. A grammar is generated using the described procedure if set (1) is consistent.

The positive set generator was introduced based on the paper by Mayer and Hamza [1]. Set creation begins with grammar conversion to linear grammar. Then, a graph based on it is created. Positive examples are created using given paths, that consists of 1,2 and 3-element combinations of rules.

The negative set is created iteratively - a random string built using terminal symbols is created and entered into the CYK algorithm [4]. If the algorithm does not parse the string, it does not belong to the language, so it is a negative example. This procedure is repeated until the demanded number of examples is created.

An example grammar generator run is printed in Table 1 for given parameters – $R_P^- = 2$, $R_P^+ = 0$, $R_B = 1$, $R_I = 2$, $S_{NT} = 4$ and $S_T = 3$. The symbols and rules added in a certain step are marked in bold. The final grammar was obtained using the latest version of tool available at the web site [3].

Future work will focus on introducing new grammar attributes that describe their behaviour in terms of example generation and structure. Consequently, that would lead to new parameter creation for the generation process, allowing thegenerated grammar's specific attributes to be easier to control and customize.

Step 1		Step2		Step 3		Step 4		Step 5		Step 6	
S	**R**	**S**	**R**	**S**	**R**	**S**	**R**	**S**	**R**	**S**	**R**
A	**A→ab**	A	A→ ab	A	A→ab	A	A→ab	A	A→ab	A	A→ab
a		**B**	**B→bc**	B	B→bc	B	B→bc	B	B→bc	B	B→bc
b		a		**C**	**C→AA**	C	C→AA	C	C→AA	**$**	$→AA
		b		a		a	**A→Cc**	a	A→Cc	a	A→$c
		c		b		b		b	**C→Bc**	b	$→Bc
				c		c		c		c	

Table 1: An example grammar generator run.

Acknowledgements

The research was supported by National Science Centre Poland (NCN), project registration no. 2016/21/B/ST6/-02158.

REFERENCES

1. Mikaël Mayer and Jad Hamza. 2016. Optimal test sets for context-free languages. *arXiv preprint arXiv:1611.06703* (2016).
2. Yasubumi Sakakibara. 2005. Learning context-free grammars using tabular representations. *Pattern Recognition* 38, 9 (2005), 1372–1383.
3. Olgierd Unold, Agnieszka Kaczmarek, and Łukasz Culer. 2018. Language Generator. `http://lukasz.culer.staff.iiar.pwr.edu.pl/gencreator.php`. (2018).
4. Daniel H Younger. 1967. Recognition and parsing of context-free languages in time n 3. *Information and control* 10, 2 (1967), 189–208.