An Enhancement to CYK Algorithm for Grammar Induction

Paweł A. Ryszawa
Military University of Technology, Faculty of Cybernetics
pawel.ryszawa@wat.edu.pl

Abstract
A novel view on grammar induction process is proposed here. It introduces some enrichments to the well-known CYK algorithm in the hope of improving its performance. The expected speed-up is believed to result from the fact that it is less costly to compute a CYK table for some grammar from another CYK table that was computed for a similar grammar.

Author Keywords
Grammar induction; CYK algorithm.

Grammatical inference has recently earned more interest as the computing power is increasing every day and new results of research in this area are being obtained. At its early stage, the research on formal grammars was focused on parsing with a formal grammar given *a priori*. That is, one might have asked: What should a correct text look like, given a grammar that is known to be correct? However, one might also raise the opposite question: What is a correct grammar given a text or texts known to be correct? While the former question has well known answers proved in the area of parsing and compiler construction, the latter shows more difficulties.

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

It is assumed that grammar productions are of one of the following forms:

$$\langle\text{nonterminal}\rangle \rightarrow \langle\text{nonterminal}\rangle\,\langle\text{nonterminal}\rangle \quad (1)$$

$$\langle\text{nonterminal}\rangle \rightarrow \langle\text{terminal}\rangle\,. \quad (2)$$

It is also assumed that the grammars do not produce empty strings, hence it is not necessary to include production rules of the form $\langle\text{nonterminal}\rangle \rightarrow \varepsilon$. Both (1) and (2) form Chomsky Normal Form for the class of grammars in question.

Assumed grammar and its encoded form

1. For every terminal symbol, there exists unique nonterminal one. The following production rules connect them and are assumed to belong to the grammar: $A_1 \rightarrow a_1, A_2 \rightarrow a_2, \ldots, A_w \rightarrow a_w$, where a_i's are terminal symbols.
2. There might be more nonterminal symbols: $V = \{A_1, \ldots, A_w, A_{w+1}, \ldots, A_p\}$.
3. There might also be more production rules - each from the following (ordered) list of p^3 rules: $P_1 : A_1 \rightarrow A_1A_1, P_2 : A_1 \rightarrow A_1A_2, \ldots, P_p : A_1 \rightarrow A_1A_p, P_{p+1} : A_1 \rightarrow A_2A_1, \ldots, P_{p^3} : A_p \rightarrow A_pA_p$
4. There is a distinguished starting symbol $S \in V$.

The grammar is represented by the binary strings of length p^3. The 1 at the n-th position means that P_n belongs to the grammar. If this is 0, then P_n is not there.

Grammar Induction as an Optimisation Problem

1. The CYK table is the very basic data structure for the algorithm:

$$\begin{array}{lllll}
X_{1,l} & & & & \\
X_{1,l-1} & X_{2,l} & & & \\
X_{1,l-2} & X_{2,l-1} & X_{3,l} & & \\
\vdots & \vdots & \vdots & \ddots & \\
X_{1,1} & X_{2,2} & X_{3,3} & \ldots & X_{l,l}
\end{array}$$

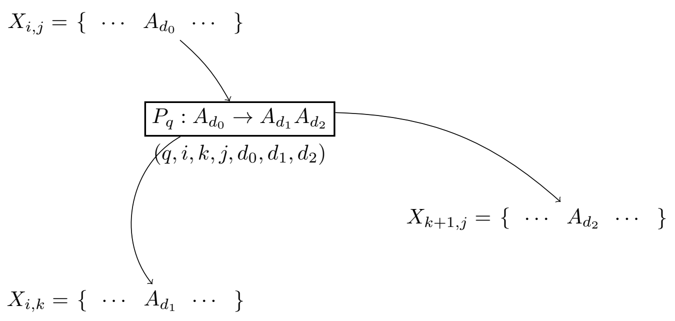
2. The algorithm is a binary optimisation problem without restrictions. Feasible solutions consist of all possible binary representations of the underlying grammars (given the assumed maximal number of production rules).
3. The fitness function prefers smaller grammars to bigger ones (the less 0's in their binary representations, the better). However, grammars producing bigger parts of the input text (positive example) are preferred to other grammars. The greatest difference between i and j, such that $X_{i,j}$ is not empty, determines the longest substring of the input text that the grammar can produce. Hence, the greater such $j - i$, the better.
4. Each iteration gives a potential solution (binary string) that do not differ much from the previous one. Thus, their CYK tables do not differ much as well.
5. Intuitively, the computing of CYK table in some iteration should make use of a previous one as both might be similar.
6. The smallest difference between two CYK tables takes place, if their underlying grammars differ by one production rule, i.e. the binary strings encoding them differ by exactly one digit. Other differences consist in having binary strings differing by more digits.

7. It is costly to compute the whole CYK table. However, it should not be that costly to compute new CYK table from a similar one.

Enhanced CYK data structure

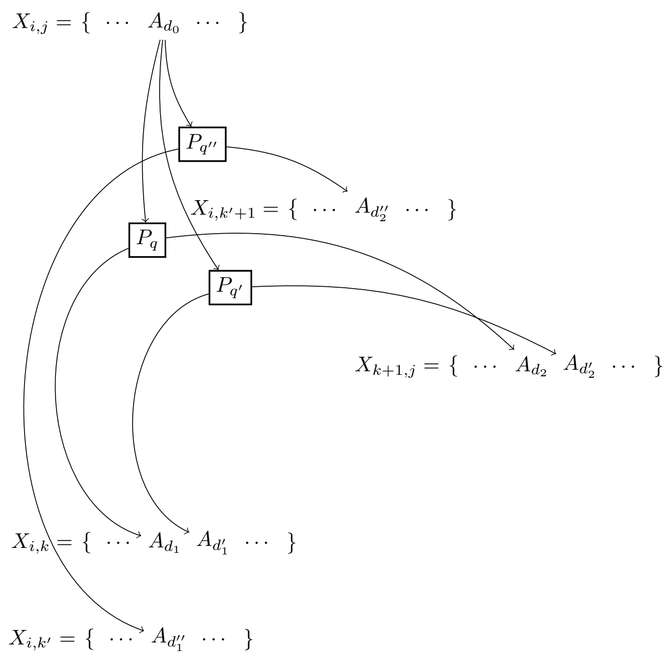
1. The CYK table needs to be enhanced to allow for easy transformation between two similar ones.

2. If, based on production rule $P_q : A_{d_0} \to A_{d_1} A_{d_2}$, the nonterminal symbol A_{d_0} was put to the set $X_{i,j}$ because A_{d_1} has already been put to some $X_{i,k}$ and A_{d_2} has already been put to some $X_{k+1,j}$, it should be remembered as a tuple of 7 numbers in an auxiliary table (e.g. in some relational database). For the sake of simplicity, the tuple can be represented by the corresponding indices: $(q, i, k, j, d_0, d_1, d_2)$. This is depicted as follows:



$X_{i,j} = \{ \cdots A_{d_0} \cdots \}$
$P_q : A_{d_0} \to A_{d_1} A_{d_2}$
$(q, i, k, j, d_0, d_1, d_2)$
$X_{k+1,j} = \{ \cdots A_{d_2} \cdots \}$
$X_{i,k} = \{ \cdots A_{d_1} \cdots \}$

3. It may happen that A_{d_0} in $X_{i,j}$ can be derived in more than one way:



$X_{i,j} = \{ \cdots A_{d_0} \cdots \}$
$P_{q''}$
$X_{i,k'+1} = \{ \cdots A_{d_2''} \cdots \}$
P_q
$P_{q'}$
$X_{k+1,j} = \{ \cdots A_{d_2} A_{d_2'} \cdots \}$
$X_{i,k} = \{ \cdots A_{d_1} A_{d_1'} \cdots \}$
$X_{i,k'} = \{ \cdots A_{d_1''} \cdots \}$

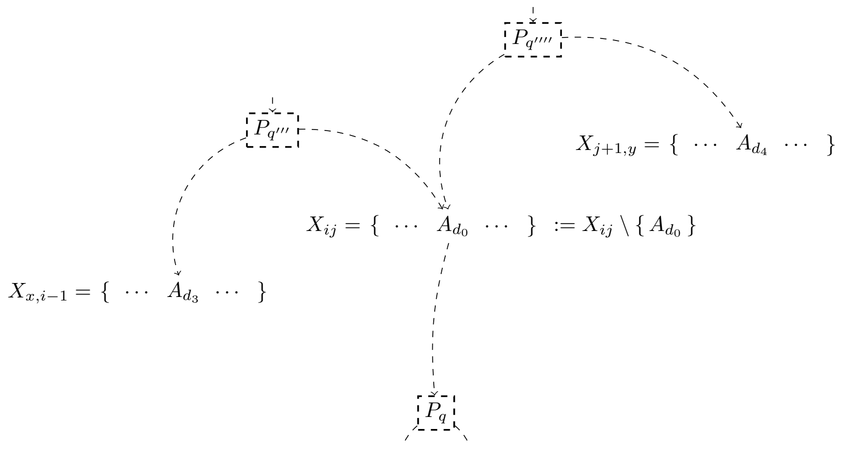
Then, more than one appropriate row should be inserted to the auxiliary table.

From one CYK table to another

1. Adding new production rule P_r to the underlying grammar means looking for derivations represented by tuples of the form $(r, *, *, *, *, *, *)$ where * stand for "any value". By cascade, finding new nonterminal symbols in sets $X_{i,k}$ and $X_{k+1,j}$, for some $i \leq k \leq j$, entails matching them against right-hand side of any existing production rule and, if necessary, add its left-hand side symbol to $X_{i,j}$, and so on...

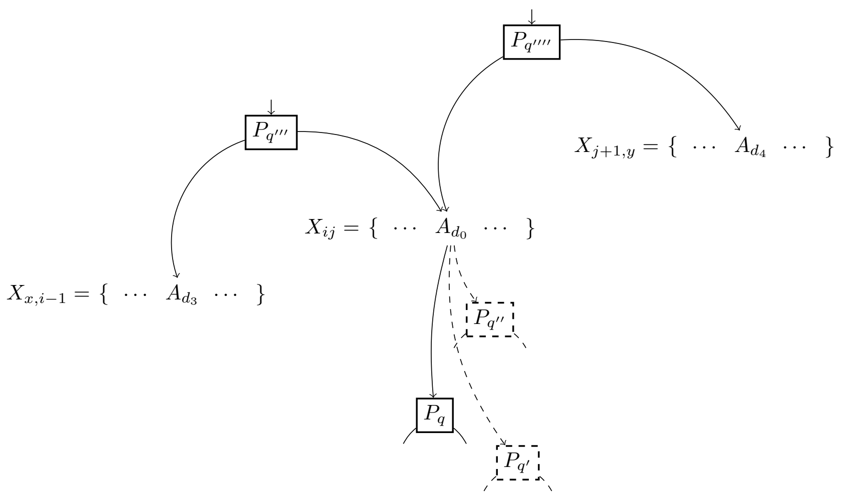
2. Deleting some production rule P_r means deleting all the rows of the form $(r, *, *, *, *, *, *)$ from the aux-

iliary table. If some derivation represented by this row was the last one to derive some symbol A_{d_0} in $X_{i,j}$, this symbol should be removed from there and, by cascade, further derivations represented by $(*,*,i,j,*,*,d_0)$ and $(*,i,j,*,*,d_0,*)$ should be removed as well, and so on... The following picture shows removed derivations (or its representations in the auxiliary table) in dashed lines:



$P_{q''''}$
$P_{q'''}$
$X_{j+1,y} = \{ \cdots A_{d_4} \cdots \}$
$X_{ij} = \{ \cdots A_{d_0} \cdots \} := X_{ij} \setminus \{ A_{d_0} \}$
$X_{x,i-1} = \{ \cdots A_{d_3} \cdots \}$
P_q

If, on the other hand, a particular derivation was not the last one to generate A_{d_0} in $X_{i,j}$, this nonterminal symbol is not removed and a cascade removal of further derivations is not triggered either.



$P_{q''''}$
$P_{q'''}$
$X_{j+1,y} = \{ \cdots A_{d_4} \cdots \}$
$X_{ij} = \{ \cdots A_{d_0} \cdots \}$
$X_{x,i-1} = \{ \cdots A_{d_3} \cdots \}$
$P_{q''}$
P_q
$P_{q'}$

3. The above procedure is expected to be faster than computing the whole CYK table from scratch just because of adding this extra production rule.

Conclusions

- Given a computed CYK table, the fitness function should be easy to compute as well.
- Having some CYK table, it is also relatively easy to transform to a CYK table for a similar grammar.
- In result, the grammar induction problem for positive examples can be expressed as a relatively fast optimisation problem.
- The algorithm should be easily implemented with SQL statements in a relational database system.

Forthcoming Research

An appropriate binary optimisation method should be applied to find a grammar in the above described method. Considered are quantum-inspired genetic algorithms (QIGA) as these are natural for unrestrictged problems. Further empirical research should focus on input texts from natural languages and QIGA as the underlying solver.

Second, negative examples play important role in grammar induction. Hence, they must be incorporated in this algorithm.

REFERENCES

1. John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2001. *Introduction to Automata Theory, Languages, and Computation* (2 ed.). Pearson Education.

2. Wojciech Wieczorek. 2017. *Grammatical Inference. Algorithms, Routines and Applications*. Springer International Publishing.

3. Daniel H. Younger. 1967. Recognition and Parsing of Context-Free Grammar in Time n^3. *Information and Control* 10, 2 (1967), 189–208.