Experiences in building of context-free grammar tree

L.. Kovács, P. Barabás
University of Miskolc/Department of Information Technology, Hungary
kovacs@iit.uni-miskolc.hu, barabas@iit.uni-miskolc.hu

***Abstract*— One of the main research areas in current intelligent systems is the development of natural language interfaces for applications. The processing of sentences is controlled by grammar of the language. From the viewpoint of efficiency, the most widely used grammar in computer linguistics is the CFG (context-free grammar) formalism. The paper provides an overview of the structure of CFG and it presents an algorithm for automatic grammar tree construction.**

I. INTRODUCTION

Computational linguistics (CL) covers the statistical and logical modeling of languages using computer-based software-hardware tools. In the 1960s, the research activities in computational linguistics were based mainly on a symbolic approach. The symbolic approach was developed out among others in the works of Chomsky [22], Harris [25] and Shannon [27]. Their methods use mainly the technologies of the parsing and artificial intelligence systems. In the shadow of symbolic approach, a smaller group of researchers stared to work out the stochastic approach. The main pioneer representatives of this direction are among others Browing [21] and Wallace [28]. The methods of the stochastic approach used mainly the Bayesian statistical algorithm to generate the language models. Although, the stochastic approach had a minor role initially, it turned out in the following decades, that it should have a dominant importance in the computational linguistics.

The first industry products of the natural language processing (NLP) systems were the language translators. The first translators worked on a word to word basis with a limited success. It is widely assumed, that the translator systems should possess metadata on the context of the text to be translated. The set of the required metadata depends on the purpose of the translation.

The language used in any NLI interfaces should meet the syntax given by a grammar. To simplify the complex behavior of the natural languages, the formal languages are used to model the natural languages. Considering the formal languages, the most widely applied grammar class is the class of context free grammars (CFG). The main benefit of CFG is the efficiency and simplicity. The usual programming languages belong to this grammar class. A special type the CFG class is the probabilistic CFG where a non-terminal symbol may have several derivations and a probability value is assigned to every possible decompositions. There are some key properties of PCFG that makes this class very important for us. It was shown that this kind of grammar can be learned from only positive examples; no negative training data is required. The second aspect is that the natural languages that are very important components in advanced human - machine interfaces can be modeled with PCFG at an acceptable accuracy.

There are only few proposals on automatic rule generation for PCFG rules. The method using TBL algorithm use a tabular representation format. The TBL (tabular representation algorithm) was proposed by Sakakibara and Kondo. Sakakibara analyzed some theoretical foundations of the algorithm. The improved TBL algorithm is not so much vulnerable to block size and population size, and is able to find the solution faster. In our project a special method was developed that uses tree alignment and adjustment operators to build up the required grammar system.

Based on the experiences it can be seen that learning PCFG from training test samples is a difficult and complex task. The best known method to build up the CFG tree is the inside-outside algorithm with an O(N3) time complexity, where N is the number of non-terminal symbols.

II. STRUCTURE OF CFG GRAMMAR TREE

The Context Free Grammar (CFG) is a tuple of four items G = (N,T,P,S), where

N. set of nonterminal symbols of the language

T : set of terminal symbols

P. set of production rules having the form $A \rightarrow \beta$, where $A \in N$, $\beta \in (N \cup T)^*$ and S is the start symbol. The language generated by G is a set L(G) of sentences where

$$L(G) = \{w \mid S \Rightarrow w\}$$

where the arrow $\Rightarrow$, defines a derivation relation. The grammar is in a Chomsky normal form if the rules are of the form $A \rightarrow \beta\alpha$, where both β, α are terminal or

978-1-4244-7430-1/11/$26.00 ©2011 IEEE

nonterminal symbols.

The test language of the investigation is the Hungarian language which has a very complex grammar. The Hungarian language belongs to the family of agglomerative languages, where a stem word can be extended with several suffixes. During the joining of suffixes the chaining of tags may cause inflection of the root part. For example, the word

kutyáimmal

can be translated into the following expression:

with my dogs

where

kutya: stem(dog)

kutyá-im : plural + genitive (my dogs)

kutyáim-mal : preposition (with)

The second difficulty of the target language is the free word order, the ordering of the words within a sentence has only few constraints. A sentence may have several equivalent forms which differ only in the order of the words. These sentences are all grammatically correct and have only slight differences in the meaning. The traditional grammar formalism like TAG, HMM are usually effective for languages with strict word ordering and with low set of acceptable words, but they are inefficient for larger size problems. The grammars like dependency grammar or word grammar are strong on handling flexible structure but their implementation details are not well explored yet.

To cope with the complexity problem within the well explored models, the probabilistic context free grammar was selected. It is known that the context free grammar is only an approximation of the natural language grammars but it provides an appropriate implementation cost. The programming languages like C or SQL all are belong to the regular languages which a special case of the CFG languages. One of the main advantages of the regular and CFG grammars is the efficient execution cost for grammar testing.

The context-free grammar can be represented with a push-down automaton. The push-down automaton is based on the LIFO processing model and has the following formal description:

P(Q,S,G,P,q,F),

where

Q: set of states

S: the alphabet of the language

G: the alphabet of the automaton

P: set of transition rules

q: initial state

F: final states.

At each phase of the sentence processing, the state of the automaton is given with a triplet (w,q,s), where w: the input sequence to be processed, q: state of the automaton, s: content of the stack.

If for a given v terminal symbol several production rules exist, the model is called probabilistic CGF model (PCGF). The main benefits of PCFG model is that it can be learned from positive data alone and it provides a robust grammar. Although the averaged entropy related to the PCFG model is higher than of n-gram models, a combination of PCGF and HMM models should superior to the traditional models.

The Generation of CFG tree

The PCFG grammar graph can be given with a graph tree representation. A relation of type $\rightarrow$ represents here a production rule, where a head is the symbol node and the tail is the rule node. A symbol may be a head in several relations. The graph graphical representation uses the following notations (see Fig 1):

non-terminal symbol node: double lined circle

terminal symbol node: single lined circle

rule node: rectangle

relation: arrow

The training set contains sentences of the form $s = w_1 \ldots w_m$, where w_i denotes a word. The generated PCFG graph should cover all training sentences. The learning algorithm adjust the PCFG graph to the training sentences. In our approach an incremental learning method is worked out as the grammar should adapt itself after every new training sentence.

The inputs for the learning algorithm are the training sentence s and the grammar graph G. In the first phase, the algorithm determines an optimal matching between the sentence and the grammar. During the matching operation, a grammar tree is generated from this compact graph. The algorithm uses a bottom-up and top-down traversing of the nodes for matching, where the top node is the S0 node.

For every node in the graph G, a fitness value ϕ is assigned. This value denotes the grade of matching, it is a value between 0 and 1. The calculation of ϕ is based on

the following considerations. Having a rule node r of the form t1,t2,..,tn and a sentence $s(w_1 \ldots w_m)$, a matching μ is defined as a partioning of the interval [1..n] into m parts. This means that a subsequence of words is assigned to every term in r. Performing the matching operation for every t_i with the assigned interval, a $\phi(t_i)$ value is generated for every symbols. The average value of the matching values is used the matching value of the mapping:

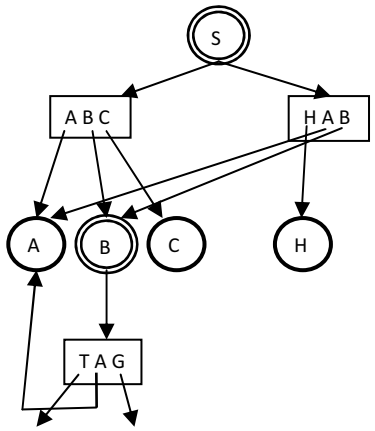

Fig 1, Sample PCFG graph

$$\phi(\mu) = 1/n \, \Sigma \, \phi(t_i)$$

As several partitioning μ can be defined between a sentence fragment and a rule, the fitness value of the node is equal to the maximum value of the mappings

$$\phi() = \max\{\phi(\mu_i)\}$$

and the winner matching is

$$\mu = \text{argmax}\, \mu \, \{\phi(\mu)\}.$$

For a non-terminal symbol node, the fitness value is calculated as

$$\phi(\mu) = \max \{ \phi(r_i)\}$$

where ri denotes a rule relation from this node. The algorithm uses also here a competitive algorithm for determining the best production rule for a symbol.

For a terminal node, the fitness value is either 1 or 0. The value 1 is given only when the interval of words contains only one word which is exactly the current terminal symbol.

After determining the winner mapping μ, the adjustment of the grammar graph is performed on the following way. The graph is traversed along the path related to the winner mapping. At every symbol nodes of this path a decision is made whether a new production rule should be added to the grammar or the best matching production rule should be adjusted to the new sentence. In this version, the number of exactly matching symbols in the rule is the decision variable:

$$\nu() = \text{count}\{t_i \mid \phi(t_i) \; < 1 \; \}$$

If this value is below a given threshold a new generation rule is created, otherwise no new rules are added. In this case only the rule symbols with inexactly matching must have a modified rule definition. In this case, the rule adjustment is passed to the child elements in a recursive way.

The generation of the best matching parse tree for a given input sentence is a crucial step from the viewpoint of execution costs. Instead of testing all the possible tree alternatives, a best-first tree construction method was implemented. The key properties of the algorithm are summarized in the following list:

1. H1 and R1 are initially empty
2. the current grammar tree is split into a pool of elementary rule items. Each rule has a head and a tail part.
3. for a given input, the rules with tail part from the input, are selected into R1
4. for every rule r in R1, the grade and position of matching is calculated
5. the compatible rules from r1 are merged into a single rule (they contain neighboring matching intervals), this new rule will replace the initial rule in R1
6. the head parts of the remaining rules in r1 are collected into H1
7. the ruleset R1 is extended with new rules containing tails from H1
8. if H1 contains the S root symbol or H1 is empty, the algorithm for best matching terminates; otherwise the algorithm goes back to point 4.
9. the grade of matching at S determines the action for update, if is lower than a threshold, the input is add as new branch to the tree, otherwise only the node with difference is modified.

The presented algorithm determines deep first algorithm to find the shortest path from a leaf to the root. As the ruleset may increase exponentially depending from the depth of the tree, a size reduction step is included in the algorithm. This reduction step removes all rule items having a low grade of matching.

Experiences

The test system uses SQL commands as training set. The goal of the test is to investigate the validity of the proposed algorithm with small positive data set. As the algorithm works on word-base, the structures of the words are not investigated. The first training set contains only a small number of examples, 200 valid SQL commands. This set is adjusted for the word-level learning, i.e.

- every expression under the sentence level are merged into a word,
- the same expressions are repeated in the samples several times.

In order to achieve a compact grammar, the nodes are assigned a mode attribute which uses the following encoding:

1: required element

?: optional element

*: multi-occurrence element.

The quality of the result depends on the training data in large scale. The training data should be optimized for the learning algorithm. If the samples are not similar to each others, no generalization is performed. With a good order of samples an acceptable ruleset can be generated as it is given in List 1

0:$S : SELECT [1] $C6 [1] $C7 [?] $C9 [?] FROM [1] $C10[1] $C2 [?] $C4 [?]
1:$C2 : WHERE [1] $C3 [1]
2:$C3 : Age>10 [1]
3:$C3 : Age<10 [1]
4:$C3 : Age=10 [1]
5:$C4 : ORDER [1] BY [1] $C5 [1]
6:$C5 : Address [1]
7:$C5 : Name [1]
8:$C5 : Age [1]
9:$C6 : Name [1]
10:$C6 : * [1]
11:$C7 : , [1] $C8 [1]
12:$C8 : Address [1]
13:$C8 : Age [1]
14:$C9 : , [1] Age [1]
15:$C10 : Person [1]
16:$C10 : Worker [1]

List 1, The generated ruleset

The Fig 2 shows that the minimum and maximum rulebase sizes in the tests in dependency from the size of the training set. In the tests, only the order of the samples was varied, the training set contained always the same samples. The figure emphases the right order in the samples.

Fig 2, Size of the generated rulebase

Another measure of the compactness of the generated ruleset is the average number of rules for a word. If a word is contained only in a few number of rules, the ruleset is compact. The execution cost of ruleset management depends significantly on this value, as the number of possible rule assignments for an input sentence is equal to the product of the rule numbers for the contained words.

Fig 3, Maximum number of rules containing the same word

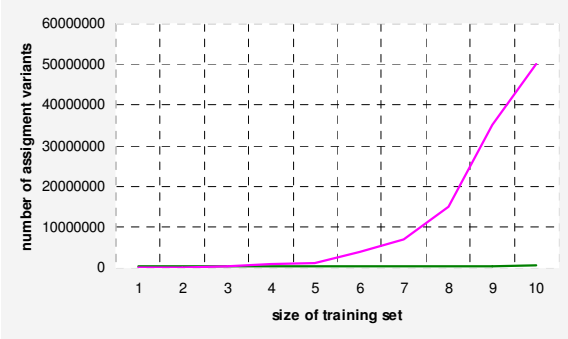

Fig 3, Number of possible assignment options

The main conclusion of the experiments was for us that the generation of a compact ruleset is possible under given conditions. The main requirements are that the training set contains the samples in appropriate order and the components of the sentence are separated at the appropriate level.

The further investigation is aimed at improving the cost values with automated ordering of the samples and with reducing the state space of possible rule assignments.

Conclusion

Using the proposed tree construction algorithm, the PCFG tree is generated for the input training samples. This unsupervised grammar generation method provides an efficient way to find out the hidden internal structures of the grammar system for any input corpus. The generated tree can be used to analyze the internal relationships of the domain objects.

Acknowledgements

The research was supported by the Hungarian National Scientific Research Fund Grant OTKA K77809.

References

[1] Jurafsky D., Martin J. H., "An introduction to speech recognition, computational linguistics and natural language processing", 2006

[2] Krenn, B., Samuelsson C., "The Linguistic's Guide to Statistics", 1997

[3] Nakamura K, Matsumoto M: "Incremantal Learning of Context Free Grammar", ICGI 2002, LNAI 2484, 2002, pp. 174-184

[4] Nakamura K, Matsumoto M: "Incremantal Learning of Context Free Grammar Based on Bottom Up Parsing and Search", Pattern Recognition 38(9), 2005, pp 1384-1392

[5] J. F. Sowa, "Semantic networks," in Encyclopedia of Artificial Intelligence, S. C. Shapiro Ed., 2nd ed., Wiley, 1992.

[6] S. Klein and R. F. Simmons, "Syntactic dependence and the computer generation of coherent discourse," *Mechanical Translation* 7, 1963.

[7] D. J. Allerton, *Valency and the English Verb*, New York: Academic Press, 1982.

[8] I. A. Mel'cuk, "Towards a linguistic ''Meaning ⇔ Text' model," in *Trends in Soviet Theoretical Linguistics*, F. Kiefer Ed., Dordrecht: Reidel, pp. 35–57, 1973.

[9] J. Steele Ed., *Meaning-Text Theory*, Ottawa: University of Ottawa Press, 1990.

[10] R. Schank Ed., *Conceptual Information Processing*, Amsterdam: North-Holland Publishing Co., 1975.

[11] R. Hudson, "Recent developments in dependency theory," in *Syntax. Ein internationales Handbuch zeitgenöössischer Forschung*, J. Jacobs, A. v. Stechow, W. Sternefeld and T. Vennemann Eds., pp. 329–338, Berlin: Walter de Gruyter, 1993.

[12] C. J. Fillmore, "The case for case," in *Universals in Linguistic Theory*, E. Bach and R. T. Harms Eds., New York: Holt, Rinehart and Winston, pp. 1–88, 1968.

[12] R. Debusmann, "Extensible Dependency Grammar: A modular grammar formalism based on multigraph description," PhD thesis, 2006.