A Statistical Approach to Parsing of Czech

Daniel Zeman
ÚFAL, MFF, Univerzita Karlova, Praha
zeman@ufal.mff.cuni.cz

The present paper is based on my dissertation that was defended at Charles University in May 1997. I introduce here a simple probabilistic model of Czech syntax, based on training data extracted from a text corpus. Then I try to employ this model in building syntactic trees of Czech sentences. In this paper, a brief description of the model is given, as well as a summary of the test results. The work is one of the first attempts to use statistical learning methods for parsing of Czech. It is to be said here that the procedures used to parse English cannot be easily ported to Czech because some specific characteristics of Czech (free word order, and rich flexion implying a huge number of word forms) cause a different behaviour of a parser. The work presented is neither final, nor satisfying solution of the problem. It rather demonstrates the primary results and proposes some basic ideas of further research.

1. Introduction

The task of parsing text (or speech) is something that a human being solves many times every day. Reading a text or listening to someone, one needs to parse each sentence if s/he wants to understand the utterance. One needs to know, for instance, that in the sentence ”The child ate the yellow apple.” the word ”yellow” combines with the word ”apple” rather than with ”child”. Wrongly parsed sentences would have significantly different meanings.

Although a native speaker is not probably aware of that, s/he actually is parsing constantly. Foreigners may have some difficulties if their mother tongues have an essentially different syntax. They may be parsing knowingly. And if we want the sentence meanings to be distinguished by a computer, e.g. in machine translation, we have to deal with syntax more than would like to.

On the other side, when writing a text, a human has to know about the syntax in order to form grammatically correct sentences. Particularly in Czech, where many syntactic relations are expressed by choosing adequate word forms, there are many opportunities to make a ”syntax error”. Today we are used to write texts in a word processor and we expect our errors to be corrected by the computer. Spell-checkers find for us any non-existing word forms we write down. But even a spell-checker will fail to recognise a syntactically wrong sentence formed by word forms that are acceptable in another context. Instead, we need a grammar checker which is another important application to employ at least some steps of a syntactic analyser.

What do we actually mean by ”parsing”? In general, it means finding the grammatical structure of a sentence. There might be several ideas what this structure looks like; however, we are not going to consider the various possible approaches, rather we will present the one we will be dealing with. At this point, let us say that in our approach the structure of a sentence is represented by a *dependency tree* where the nodes are the words from the sentence, and the edges are dependencies between them. (A more formal definition of the structure will be given later.) So, we can say that the task of parsing involves constructing a dependency tree, given a string of word forms, constituting a sentence.

With statistical modelling, we will look at parsing as a random process, and we will try to predict its behaviour. We will have at our disposal a sample of previous the output of the

process, which represents an incomplete knowledge about the process. Our goal will be to analyse this sample statistically and to construct a model — a probabilistic distribution allowing us to say, which dependency tree is the most probable for a given plain sentence.

In Section 2 we give a formal definition of the parsing problem. We define the model in Section 3, which is a theoretical background; then, in Section 4 we describe the algorithm used to build the model and to employ it in parsing. In Section 5 we introduce methods of testing the model, and we present the results of the experiments. Finally, in Section 6, we summarise the results and discuss several ideas about improvements and directions of further research.

2. Sentences and Dependency Trees

A **sentence** is a sequence of words $s_1, s_2, \mathrm{K}, s_n$. Generally we do not need to know what is a word. It can be simply the word form which appeared in the sentence but it might also contain some morphological information, the lemma and so on. Additional symbols such as numbers and interpunction are considered separate words.

A **dependency tree** is formed by a set of nodes *V* and a set of edges *H*. Each node corresponds to a word from the sentence, except for the root of the tree. Each edge represents a **dependency** between a **governor** and its **subordinate** constituents. The definition of a tree requires that there is exactly one node superior to any constituent of the sentence. However, every node may have any number of subordinate constituents. The root of the tree is identified by a special node #. It does not correspond to any word from the sentence, it only serves as the root marker. Each tree has exactly one # node. The nodes are ordered so that we can define depth-first and width-first tree searches.

Note that we have not defined any rule saying how a tree is constructed given a sentence. Rather we are hoping that our statistical model discovers those rules and describes them in terms of probability. So our intuitive knowledge of how a tree is constructed will be used only indirectly, as we will define the model.

3. The Statistical Model

Let *S* be a sentence of *n* words. According to our definition, the dependency tree of that sentence will consist of $n+1$ nodes (including the root #) and *n* edges. There are $(n+1)^{n-1}$ such trees. Among them, we want to select the most probable one, i.e. the tree that a human would assign to the given sentence most often. Formally, we are looking for a tree *M*, defined as

$$\arg\max_M \; p(M|S)$$

From the Bayes' rule we get:

$$p(M|S) = \frac{p(S|M) \cdot p(M)}{p(S)}$$

Here $p(S)$ is only a constant that has no influence on the result:

$$\arg\max_M \frac{p(S|M)\cdot p(M)}{p(S)} = \arg\max_M \left(p(S|M)\cdot p(M)\right)$$

The $p(S|M)$ is the probability that the tree *M* was constructed exactly from the sentence *S*. Since we are able to get back the original sentence from the tree (e.g. by having preserved the original word forms and original word order as an additional info to each node), we understand the $p(S|M)$ to be only a technical tool. It is a binary function giving one if the tree is correct and corresponding to *S*, and zero otherwise.

The real strength then resides in $p(M)$, the probability of the tree. Unfortunately we are not able to model it directly: since there is such a big number of trees, we cannot get a parsing sample large enough to estimate their probabilities. We are forced to simplify the task essentially and to approximate the probabilities in some way, which on the other hand cannot be done without a loss of accuracy.

First, we reduce $p(M)$ to the probability $p(H)$ of an edge sequence *H*. We can do that because every tree is well-defined by the set of its edges. There is a small problem with the fact that not every sequence of edges on a given set of nodes is a tree. Nonetheless, we can define a special tree $M_{\#}$ that does not correspond to any real sentence and assign it to all wrong edge sequences. We can think of this tree as of an empty tree, containing only the root. It is clear that $p(S|M_{\#})$ will be always zero. We see that if the probability of the tree is replaced by the probability of an edge sequence, the most probable tree will be just $M_{\#}$. However, the proportions among the other trees are left intact, so that we have not distorted the results so far.

We can break $p(H)$ down as follows:

$$p(H) = p(h_1)\cdot p(h_2|h_1)\cdot\mathrm{K}\cdot p(h_n|h_1,\mathrm{K},h_{n-1})$$

Now it is time to reduce the complexity and the number of parameters of our model. We cannot expect that our sample (training) data will make it possible to us to estimate probabilities that are conditional in more than one dimensions. The longer vectors of edges we consider, the lower is the **recall** (the fewer times we find a vector in the training sample). On the other hand, if we reduce the vectors to one member, getting unconditional probabilities, we lose significantly on the **precision**. Let us have a look at the unconditional, so-called **unigram** language model. We suppose that for every edge h_i it holds that

$$p(h_i|h_1,\mathrm{K},h_{i-1}) = p(h_i)$$

Then we see that

$$p(H) = \prod_{i=1}^{n} p(h_i)$$

Now the problem is much more simple because we can estimate the edge probabilities $p(h_i)$ by the relative frequencies from the training data, $\tilde{p}(h_i)$. Since we suspect that the unigram model is insufficiently precise, we will also study the **bigram** model where

$$p(h_i|h_1,\mathrm{K},h_{i-1}) = p(h_i|h_{i-1})$$

and

$$p(H) = p(h_1) \cdot p(h_2|h_1) \cdot \ldots \cdot p(h_n|h_{n-1}) = p(h_1) \cdot \prod_{i=2}^{n} p(h_i|h_{i-1})$$

If we have a large sample of training data, we shall be able to estimate probabilities of pairs of edges, too. However, we face here an additional problem, the one of specifying the order of the edges. If we want to make the probability conditional on the preceding edge, we have to specify which edge precedes the given one. This is why we have required that the nodes of the dependency tree are ordered and that we can define depth-first or width-first search. Now we can assign the same order to the edges since each node (except of the root) has exactly one input edge. We do not know whether there is a stronger correlation between depth-first neighbours, or between the width-first ones, and whether there is any correlation at all. So we are going to try both orders in our experiments.

Finally we want to emphasise one aspect of the above language model definition. We transposed the probability of a tree to the probability of an edge sequence, where the word *sequence* implies that a predefined edge order exists. It means that among two edge permutations, at most one may be considered correct, while the other is assigned the special tree $M_{\#}$.

4. The Algorithms

The procedure will consist of two phases, one of training and another of parsing. During the training period we will go through the training data, read dependency trees and count frequencies of all the edges or edge pairs, respectively. After this phase, we are able to estimate their unigram probabilities as

$$p(h) = \tilde{p}(h) = \frac{c(h)}{c}$$

where $c(h)$ is the number of all occurrences of the edge h, and c is the number of all occurrences of all edges in the training data sample. Similarly, we can estimate the bigram probabilities as

$$p(h_i|h_{i-1}) = \tilde{p}(h_i|h_{i-1}) = \frac{c(h_{i-1}, h_i)}{c(h_{i-1})}$$

where $c(h_{i-1}, h_i)$ is the number of all occurrences of the edge pair h_{i-1}, h_i (in the given order).

The trained probability distribution has one essential disadvantage: it is very sparse: especially with the bigram model there are many edge pairs that have not occurred in the training data and so they get a probability of zero. But we should not simply forbid such pairs because once we might want to use the language model together with an additional source of information. If this additional info absolutely prefers one of those 'unseen' pairs, and that pair has a probability slighter higher than zero, it still can be selected. That is why we should avoid the zero probabilities. We have to **smooth** the distribution so that it does not contain any zero holes. With smoothing we will gain an additional advantage: we will be able to distinguish more subtly the edges that were seen in the training sample in a context different from that of the test data.

We do that by adding the unigram probabilities to the bigram ones. Albeit we have not yet seen the edge pair h_{i-1}, h_i, it is still possible that we have seen the edge h_i alone. However, if we have not seen the edge h_i at all, it would get again the zero probability. To avoid this case,

we add also a constant, the uniform probability $\tilde{p} = \frac{1}{|L|^2}$ where $|L|$ is the size of the lexicon and thus $|L|^2$ is the number of possible edges. Since the sum of three probabilities is not itself a probability (it can exceed 1), we have to weight each component by a λ from the interval $(0;1)$; the sum of all weights must be 1:

$$p(h_i|h_{i-1}) = \lambda_2 \tilde{p}(h_i|h_{i-1}) + \lambda_1 \tilde{p}(h_i) + \lambda_0 \tilde{p}$$

In our experiments we use estimated values of weights. With bigram models we use $\lambda_2 = 0{,}99$, $\lambda_1 = 0{,}009$, and $\lambda_0 = 0{,}001$. With the unigram model we use $\lambda_1 = 0{,}99$, and $\lambda_0 = 0{,}01$.

The syntactic analysis proper is performed in the second phase as we construct the dependency tree. Here we use the trained and smoothed probability distribution to find the best, i.e. the most probable tree. We will construct the tree using the greedy algorithm. We start with a discrete graph, containing the root only. Then, in each step we add the edge that increases the tree probability at most. Note that not all remaining edges are allowed to be added in each step, but only those that will not violate the correctness of the tree. Particularly, we are allowed to add those edges whose superior word has already been added to the tree, and the subordinated one has not. With the bigram models, we additionally have to reduce the allowable set to those edges that keep the given search order. By that we guarantee that the "guard" probability $p(S|M)$ will be equal to 1. Note that the greedy algorithm does not guarantee that $p(M)$ is the highest possible value. Rather it is an approximation we use to reduce the computational complexity.

We have not specified yet how the training data that we have at our disposal look like. In particular, we must specify what we understand under a "word". It might be a lexical unit (lemma), a word form, a tag containing some morphological information, or any combination of the above. Since the available data resources are quite poor at the time (April 1997) we give up training based on lexical information and concentrate ourselves on the morphological tags. At the moment, we work with resources in which the tags are not disambiguated, so that most of the word forms have a set of tags associated with them. Exactly one of the elements of the set is correct in the given context but we do not know which one it is. That is why we have to deal with whole tag sets.

It is the matter of course that both the training and the parsing phases must be amended when training on disambiguated sets of tags. If there is an edge in the training sample whose governing node contains *i* and subordinate node contains *j* tags then we have to record all $i \times j$ possible combinations (tag edges). We have to remember that these combinations represent only one edge occurrence in the training data. In order to keep the relation to frequencies of the other edges, we distribute the occurrence among the particular tag combinations. E.g., if there is an edge with the superior node `[solí, NFS7A|NFP2A|VPX3A]` and the subordinated node `[bílou, AFS41A|AFS71A]` then that edge actually represents exactly one of the combinations `NFS7A-AFS41A`, `NFS7A-AFS71A`, `NFP2A-AFS41A`, `NFP2A-AFS71A`, `VPX3A-AFS41A`, and `VPX3A-AFS71A`. Instead of incrementing frequency of some edge by one, we raise the frequencies of each combination by $\frac{1}{6}$. Then, during the parsing phase, we compute the probability of an event disjunction:

$$p(h_1 \vee \mathrm{K} \vee h_k) = p(h_1) + \mathrm{K} + p(h_k) - p(h_1 \wedge \mathrm{K} \wedge h_k)$$

The last equation member is equal to zero if no tag combination occurs in one edge more than once. It is true that in our data such a repetition is possible, but the reasons are not relevant for our task. Therefore we record every combination only once.

In the same view we have to deal with the context edges in bigram models. If the edge h_i contains the tag combinations $e_{i,1}$ to e_{i,k_i} and its context edge h_{i-1} contains the combinations $e_{i-1,1}$ to $e_{i-1,k_{i-1}}$ then, in the training phase, we record each combination from the edge in scope gradually in contexts of all combinations from the context edge, each time with the frequency $\frac{1}{k_i \cdot k_{i-1}}$. In the parsing phase, we have to remember how the probabilities are estimated by the frequencies. Because every edge frequency is actually a sum of frequencies of tag combinations, we get:

$$p(h_i | h_{i-1}) = \frac{\sum_{j=1}^{k_i} \sum_{l=1}^{k_{i-1}} c(e_{i,j} \wedge e_{i-1,l})}{\sum_{l=1}^{k_{i-1}} c(e_{i-1,l})}$$

We see that the ambiguity of morphological tags increases enormously the computational complexity, especially with the bigram distribution. Since certain words can have up to 27 possible tags, we may need about half a million accesses to the table to estimate one probability value!

5. Testing and Results

We have at our disposal sample data containing 1150 trees. We reserve 100 of them for testing purposes because the model cannot be tested on the same data it has been trained by. The rest 1050 trees (18015 words) form our training sample.

Table 1. Basic characteristics of particular models. *T* is the number of training trees. By ”param’s” we mean the number of different edges (or edge pairs) that occurred in the training sample. The value is higher than the number of words with the unigram model, because some edges are labelled by sets of tag combinations. The entropy of a model is defined by the equation

$$H(\alpha) = -\sum_{i=1}^{|L|^2} p(h_i) \log_2 p(h_i)$$

for the unigram model and

$$H(\alpha|\beta) = \sum_{i=1}^{|L|^2} p(h_i) H(\alpha|h_i) = -\sum_{i=1}^{|L|^2} \sum_{j=1}^{|L|^2} p(h_i) p(h_j|h_i) \log_2 p(h_j|h_i) = -\sum_{i=1}^{|L|^2} \sum_{j=1}^{|L|^2} p(h_i, h_j) \log_2 p(h_j|h_i)$$

for the bigram model. The perplexity is $P(\alpha) = 2^{H(\alpha)}$.

Basic model characteristics	**words**	**param's**	**entropy**	**perplexity**
unigram, T=50	684	3127	9,43	689
unigram, T=250	3960	10 305	10,64	1599
unigram, T=1050	18 015	23 099	11,12	2227
bigram depth-first, T=50	684	78 026	2,51	6
bigram depth-first, T=250	3960	439 964	3,64	13
bigram depth-first, T=1050	18 015	1 901 858		
bigram width-first, T=50	684	51 775	2,70	7
bigram width-first, T=250	3960	456 723	3,88	15
bigram width-first, T=1050	18 015	1 943 864		

We tested each model in the following way: we took a tree from the test data, reconstructed the plain sentence (i.e. removed the tree structure) and made it the input of the parsing procedure. After parsing, we thus had two trees, the old one M_0 from the data, and the new one M_1, generated by the parser. Then we counted the wrong edges, i.e. the number of words that were subordinated to different nodes in both trees. We defined the **success rate** as the number of all wrong edges in the test output, divided by the number of all edges in the test data. Note that for this purpose we compared the word occurrences rather than the contents of the tag sets. Hence if two words had the same tag sets and the parser swapped their positions in the tree, they both were counted as wrong edges.

Table 2. Again, *T* is the number of the training trees. *Z* is the number of the test trees.

Success rate	**T = 50 Z = 5**	**T = 250 Z = 25**	**T = 1050 Z = 100**
unigram	25 %	32 %	31 %
bigram depth-first	29 %	22 %	30 %
bigram width-first	23 %	23 %	27 %

Table 3. The cross entropy is defined as follows:

$$\hat{H}(\alpha) = -\sum_{i=1}^{|L|^2} \tilde{p}(h_i) \log_2 p(h_i)$$

for the unigram model and

$$\hat{H}(\alpha|\beta) = \sum_{i=1}^{|L|^2} \tilde{p}(h_i) \hat{H}(\alpha|h_i) = -\sum_{i=1}^{|L|^2} \sum_{j=1}^{|L|^2} \tilde{p}(h_i) \tilde{p}(h_j|h_i) \log_2 p(h_j|h_i) = -\sum_{i=1}^{|L|^2} \sum_{j=1}^{|L|^2} \tilde{p}(h_i, h_j) \log_2 p(h_j|h_i)$$

for the bigram model. As before, the symbol $\tilde{p}$ denotes the relative frequency in the test data rather than probability. The higher the cross entropy is, the more edges of the test data are not known from the training data.

Cross entropy	**T = 50 Z = 5**	**T = 250 Z = 25**	**T = 1050 Z = 100**
unigram	21,68	14,84	12,60
bigram depth-first	21,98	10,81	
bigram width-first	21,45	11,11	

We see that the results are not very satisfactory. It is surprising that the error rate with bigrams is higher than that with unigrams. Most likely insufficient data resources cause this. However, it is also possible that the defined context has a minor influence in the parsing process. The disadvantage of limiting allowed edges by depth-first or width-first search order then comes to the foreground.

Our parser is only slightly more successful than a procedure that assigns trees to sentences in a certain pre-defined way. After this procedure, each word depends on the preceding word in the sentence, only the first word and the final period depend on the root. We do not present any comparison here because in the experiments the error rate was measured in a different way in order to compare it to the work of Ribarov (1996). For details, see Zeman (1997) and Ribarov (1996).

An important question is whether the errors of our method are caused mostly by concessions made in the model definition or by an inaccurate construction of the most probable tree. If the model (together with training data) is accurate, the original trees from the data will get higher probabilities than the wrong trees generated by the parser. We have demonstrated experimentally that this is the weakness of our method, see the next table.

Table 4. The values N, and N_0 are the numbers of tested trees, and the numbers of cases when the generated tree received a higher probability than the original one, respectively. We also define the **difference of two trees** as

$$d(M, M_0) = \overline{p}(M) - \overline{p}(M_0)$$

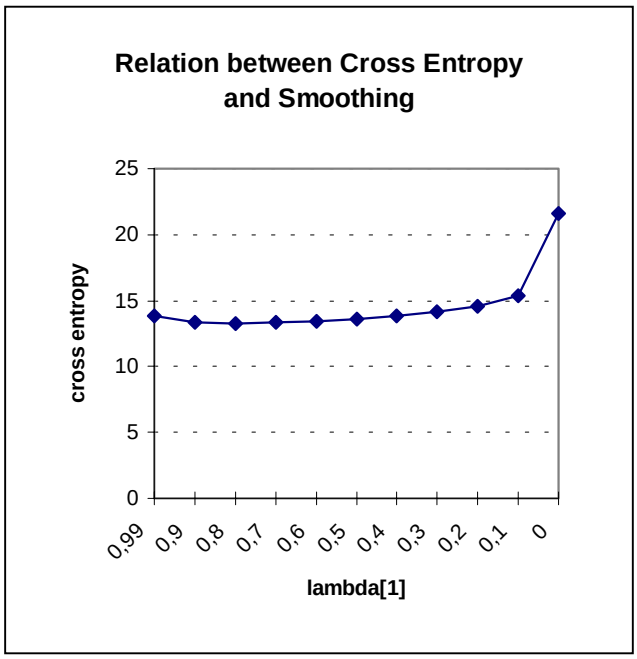
where $\overline{p}(M)$ is the probability of the tree *M*, normalised to the number of nodes as follows.

$$\overline{p}(M) = \prod_{i=1}^{n} \sqrt[n]{p(h_i)}$$

Actually it is a geometrical average of edge probabilities. Note that a positive difference indicates the generated tree being more probable, and a negative one indicates the opposite. In Table 4 we present *average* differences.

Average probability difference between generated and original tree	**N**	**N_0**	**$d(M,M_0)$**
unigram, T = 50	5	5	$+6{,}0.10^{-3}$
unigram, T = 250	25	22	$+2{,}9.10^{-3}$
unigram, T = 1050	100	89	$+2{,}8.10^{-3}$
bigram depth-first, T = 50	5	5	$+1{,}8.10^{-3}$
bigram width-first, T = 50	5	5	$+2{,}0.10^{-3}$

Table 5. The last experiment will show the influence of the smoothing weights λ_i. If it appeared significant, we could specify the weights by the EM algorithm instead of estimating them. We experimented with an unigram model, trained on 250 and tested on 25 sentences. Gradually we dropped the value of λ_1 and raised λ_0. We have demonstrated that there is almost no influence on the error rate and, as the figure shows, only a minor influence on the cross entropy.



6. Summary

With the data available in April 1997, none of the tested methods seems to be useful for parsing Czech. Our observations point to a weakness of the probability distribution rather than to a bad edge searching (the greedy algorithm). This does not mean, however, that the statistical approach should be completely thrown out. We have made several simplifications and approximations, forced partly by the time and space computational complexity, partly just by the amount and quality of available data. Let us summarise now some points of possible improvements, which may therefore become an object of further research. (The order of items does not necessarily correspond to their importance.)

1. The model uses dependencies between morphological tags but ignores the relations between lexical units. For example, a comma in the sentence is marked by the same tag as any other punctuation mark, although it surely behaves in a different way than, e.g., the final full stop. An ”ideal” model should take into account a combination of lexical as well as morphological relations between the governing and dependent nodes, and, if possible, take at least an advantage of bigrams.
2. The fact that one node can have several variants of morphological tags is most likely a significant source of inaccuracy. For instance, the Czech word form ”V” may be not only a preposition at the beginning of the sentence but also a Roman number symbol 5; however, in our model both derived edge groups receive the same weight. We can list many defects of this type. Very often there are several combinations of number and case that can be assigned to a substantive or adjective word form. Exactly these attributes play a significant role in finding dependencies (the agreement between a noun and its attribute or between the predicate and the subject). We believe that as soon as adisambiguated morphological analysis (or tagging) is available, the precision will increase. At the same time the complexity of the computation should fall down.
3. In some cases we might use less and in other cases more specialised tags. For example, the difference between particular pronoun types should not have major influence on the syntax while their gender, number and case are quite important. On the other hand there is an essential difference between a comma and the final full stop, but in our data the same code `ZIP` is assigned to both of them (see also point 1).
4. The smoothing weights were estimated. As shown in Table 5, under current circumstances this approximation makes no difference to the success rate of the model. However, certain

changes of cross entropy in the same Table mark that with much more data we can expect an increasing influence of the smoothing weights. Then we could improve the result by training the weights with some variant of the EM algorithm.

5. We have not used all the information we had about the sentence. In particular, we should consider the word order, suppress the non-projective edges, and find a suitable way to use the distance between the superior and the subordinate nodes in the sentence.

6. In Ribarov (1996), the tree is not constructed from an empty one. Instead, the sentence is deterministically structured to a start tree where each word depends on the preceding one, only the first word and the final full stop depend on the root. Then the start tree is adjusted as long as there is an edge improving probability. It might be interesting to change our method this way; however this probably will not improve anything before we construct a better probability distribution.

7. The training data contained errors — that is, errors with respect to the rules a human uses to parse a sentence, because otherwise the ”truth” is defined just by the contents of the training sample. If these mistakes appear with the same frequency in the test data, their existence shall not hinder. (Examples: badly placed end of the sentence; sentence parsed as a chain where each word depends on the preceding one.)

7. Acknowledgements

The research would not be possible without a corpus of manually annotated sentences (the training data). The (yet incomplete) corpus we used has been made available by the grant GAČR No. 405/96/K214 and by the project of the Ministry of Education of the Czech Republic No. VS96151.

8. References

[1] [BAHL ET AL. 1989]
Lalit Bahl, Peter Brown, Peter de Souza, Robert Mercer: *A Tree-Based Statistical Language Model for Natural Language Speech Recognition*. In: IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, No. 7, July 1989. Yorktown Heights.

[2] [BROWN ET AL. 1993]
Peter Brown, Stephen della Pietra, Vincent della Pietra, Robert Mercer: *The Mathematics of Machine Translation: Parameter Estimation.* In: Computational Linguistics, 19 (2): 1993, pp. 263-312.

[3] [COVER, THOMAS 1991]
Thomas Cover, Joy Thomas: *Elements of Information Theory.* John Wiley & Sons, Inc., New York 1991.

[4] [DEMPSTER ET AL. 1976]
Arthur Dempster, N. M. Laird, Donald Rubin: *Maximum Likelihood from Incomplete Data via the EM Algorithm.* In: Journal of the Royal Statistical Society, Series B, Volume 39, London 1977, pp. 1-38..

[5] [HOLAN ET AL. 1995]
Tomáš Holan, Vladislav Kuboň, Martin Plátek: *An Implementation of Syntactic Analysis of Czech.* In: Fourth International Workshop on Parsing Technologies. Praha, 1995.

[6] [CHOMSKY 1957]
Noam Chomsky: *Syntactic Structures.* Mouton & Co. Publishers, Den Haag 1957.

[7] [OLIVA 1989]
Karel Oliva: *A Parser for Czech Implemented in Systems Q.* Explizite Beschreibung der Sprache und automatische Textverarbeitung XVI. Matematicko-fyzikální fakulta Univerzity Karlovy, Praha 1989.

[8] [PETKEVIČ 1998]
Vladimír Petkevič: *Underlying Structure of Sentence Based on Dependency*. Filozofická fakulta Univerzity Karlovy, Praha *(in press)*.

[9] [RIBAROV 1996]
Kiril Ribarov: *Automatická tvorba gramatiky přirozeného jazyka (diploma thesis)*. Matematicko-fyzikální fakulta Univerzity Karlovy, Praha 1996.

[10] [SGALL ET AL. 1969]
Petr Sgall, Ladislav Nebeský, Alla Goralčíková, Eva Hajičová: *A Functional Approach to Syntax in Generative Description of Language.* American Elsevier, New York 1969.

[11] [ZEMAN 1997]
Daniel Zeman: *Pravděpodobnostní model významových zápisů vět (diploma thesis)*. Matematicko-fyzikální fakulta Univerzity Karlovy, Praha 1997.