Faculty of Mathematics and Physics
Charles University

AUTOMATIC BUILDING OF A DEPENDENCY TREE

—THE RULE-BASED APPROACH AND BEYOND

Kiril Ribarov

PhD Thesis

Prague - 2004

To Monia, Tereza and Luca

Contents

Foreword 7

1. Introductory Notes and Past Work **11**
1.1. The origins of this work 11
1.2. First modifications of RBA 13
1.3. On the word parser 18
1.4. The theoretical framework and automatic learning 19
1.5. On analytical and tectogrammatical level 22
1.6. Early parsers and processing of Czech 25
1.7. Current trials on parsing of Czech 28
1.8. Breaking the problem 29

2. Some Aspects of PDT **33**
2.1. Using PDT 33
2.1.1. Node values of interest 34
2.2. Basic facts 34
2.3. Saturation of PDT 37

3. Further Modifications of the Rule-Based Approach for Czech **39**
3.1. The input 39
3.2. Using modified tagsets 40
3.3. Linguistic recommendations for morphemic tagset modifications for syntactic parsing 44
3.4. On rule modifications 45
3.5. The error function 49
3.6. Learning discipline 51
3.7. Success and conclusion on the RBA modifications 51
3.8. Tagging with analytical functions 54
3.8.1. Conclusion 55
3.9. Summary on the RBA modifications 56
3.9.1. The percentages of RBA 57

4. The Directed Minimum Spanning Tree **59**

5. Naive Parsing **63**
5.1. The grammar and sentence graphs 63
5.2. MST as a dependency tree finder using frequencies 64
5.3. The frequency MST parser 65
5.4. MST as a dependency tree finder using dependency pictures 69
5.5. The percentages of naive parsing 73

6. À la RBA 77
6.1. Learning improvement 77
6.2. The rules 78
6.2.1. The Simple rule 78
6.2.2. The Concord rule 79
6.2.3. The Distance rule 79
6.2.4. The Between rule 79
6.2.5. The Not-Between rule 79
6.2.6. Other rules 79
6.3. Examples of several rule instances 80
6.4. Observations, characteristics and results 81
6.5. Definition of a second phase - tree modifications 82
6.5.1. Preceding dependency 83
6.5.2. Brother dependency 83
6.6. A comment 84
6.7. The percentages of à la RBA 84

7. The Perceptron-Based Approach for Building of a Dependency Tree 87
7.1. The perceptron algorithm for tagging 87
7.2. Formalizing the features 89
7.2.1. Tagging 89
7.2.2. Dependency parsing 89
7.2.3. Global representations 89
7.3. Score 90
7.4. Perceptron-based dependency parsing algorithm 90
7.4.1. Learning 90
7.4.2. Application 91
7.5. Feature types 92
7.5.1. Tests with additional feature types 93
7.6. Results 94
7.7. Considerations for better features and algorithm performance 96
7.8. The percentages of the perceptron-based approach 97

8. Get Almost Two For the Price of One 99

9. Final Notes 103

References 107

Appendix 111

Foreword

As many works during the past decades, also this study wants to contribute towards a more successful automatic parsing of a natural language. My aim is to present new frameworks, which I support by program tools and, where of interest, I present result evaluations. All approaches discussed are based on a dependency syntax framework, but not necessarily limited only to it. It is my aim neither to present a final product, nor to present a manual to a ready-to-use best parser.

As it has resulted from many discussions during the past years between linguists and computer scientists (all of them computational linguists) I feel the need for stressing the following: the approaches studied in this work are only those with which an automatic acquisition of the surface syntactic structure of the natural language (in a form of a list of rules, features, parameters) is possible. Such, automatically obtained schemes are applied and tested. This contribution does not study formal humanly crafted grammars (being successful but not excellent in parsing, e.g. grammars coded in a form of rewriting rules or similar).

The automatically obtained results presented here, if not being stated otherwise, are not filtered, neither manually purified nor modified in any other way, since the main topic of interest is to find out how these methods intrinsically perform.

The theoretical field of formal studies of the language system mainly for Czech has a long (esp. Praguian) tradition, being an example of a deep and vast and more improving than changing theory. Although, natural language processing algorithms presented here are so-called language independent, one cannot isolate completely the language from its analysis. The primary language of interest is Czech and the theory to fit is the dependency syntax as described in (Sgall, Hajičová, Panevová 1986).

There are more past works discussing problems of generation of a Czech sentence than of its analysis. Nevertheless, one will not remain barefoot even within the analysis of Czech mainly thanks to the last eight years (the creation of the Prague Dependency Treebank, a period of exhaustive syntactic analyses) including also significant results of the last decades. Projects like TIBAQ or RUSLAN did incorporate parsing modules using hand crafted rules and a

restricted dictionary. Unfortunately, it is still not possible to combine effectively the past manual work with the current automatic one.

The Prague Dependency Treebank constitutes a clear interface line between a needed algorithm and the linguistic knowledge coded in an appropriate theoretical framework: it is the linguist who guarantees the correctness of the manual annotation i.e. linguistic information added to the text in the language corpus and it is the linguist who 'meta' defines the behavior of the language; it is the computer scientist who extracts the linguistic information from the corpus and is able to design algorithms which represent it, generalize it, use it to all its extends, play with it; it is the treebank, the annotated corpus, which is the interface/communication channel to the outer non-linguistic world. Adding the character of the language to process, this is the technical core of deriving function from form.

None of the methods presented here, use other than treebank information. As always, the extra, proper linguistic, knowledge is implicitly included in any automatic model by the design of the model itself, e.g. creation of rule templates within the rule-based approach, which is the central approach in this study, or in the postulation of how *n*-gram is defined within statistical language modeling.

This work presents the following algorithms for automatic acquisition of analytical trees (not only) for Czech:

- the modified rule-based approach (Chapters 1, 3 and 6),
- the modified perceptron-based approach (Chapter 7),
- a naive, frequency-based approach (Chapter 5),
- a naive, so-called picture-based approach (Chapter 5),
- a definition of the parsing-by-tagging approach (Chapter 8).

For each of the mentioned approaches there are programs created in C (the older experiments) and in Perl (the more recent experiments) and those are available for the purpose of academic research upon request to the author. The terms success rate or precision are used in this work as synonyms and if not otherwise stated they mean dependency accuracy, i.e. number of correct dependencies over all tree dependencies.

I hope that the mentioned algorithms will be understood as steps forward in the possibilities for natural language processing - the currently most successful algorithms for automatic assignment of an analytical tree structures for Czech remain to be the Michael Collin's parser followed by the Eugene Charniak's one achieving 84% success rate. The presented algorithms propose original and innovative solutions to the parsing problem. Due to the new elements they present it was not always possible to approach to a detailed evaluation or their further extensions or further modifications. It seemed obvious therefore to concentrate more on the algorithms themselves and their basic application. Except for the first three chapters, this work is presented here for the first time. If while reading, it has provoked some ideas you believe are interesting and even such that are not explicitly stated here, I will assume that this thesis has fulfilled its aim.

I owe the chance to perform the studies presented in this work to Eva Hajičová and Jan Hajič for the working conditions they have created by the LPZJD grant and currently by the Center for Computational Linguistics. I would also like to thank Jan Hajič for presenting me the work of Eric Brill and Michael Collins. I would like to thank Petr Pajas for creating such a wonderful tool for traversing treebank trees as the TrEd is, specially its batch variant bTrEd.

Thank would be a weak word to express my gratitude to Alena and Bára who have always been close to me and have shared all my moments with difficulties and moments of happiness.

As the time passes I realize that I am more and more grateful and, in fact, truly happy, to have been and still be close to such extraordinary people as Petr Sgall, Jarmila Panevová and Eva Hajičová are, who have given me a professional background, encouragement and light.

Kiril Ribarov
Prague - July, 2004

1. Introductory Notes and Past Work

1.1. The origins of this work

In my master thesis (Ribarov 1996) and as later published in (Hajič, Ribarov 1997) for a first time an attempt at an application of a rule-based approach originally proposed by Eric Brill (Brill 1992, 1993a,b,c), for an automatic grammar generation of Czech was demonstrated. The mentioned work grew in a time when no treebank of Czech was available. A small data bank had to be created to allow for testing the algorithm and building and analyzing its modifications in order to be able to apply it for a dependency syntactic framework of Czech. Since these results are important for the later development and improvements, I summarize here what has been done in the above mentioned contributions. At the same time I use this occasion to introduce the foundations of the rule-based approach and the used terminology.

Belonging to the group of algorithms for automatic natural language learning the *rule-based approach* (RBA) applied to *parsing* can be summarized in the following diagrams, one for the process of application used to parse an input text when a set of rules exists (Figure 1.1), and the other one for the process of learning during which the *set of rules* is created (Figure 1.2).

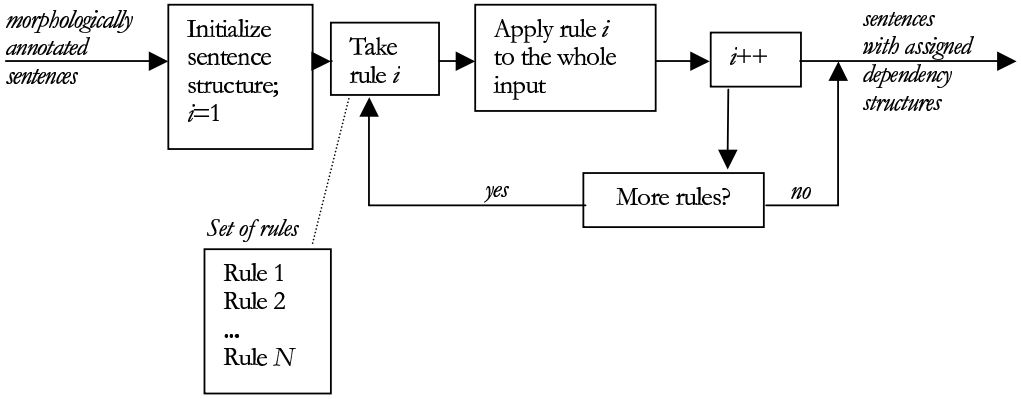


Figure 1.1: RBA - Application

The rules are obtained automatically, although one may imagine also a set of manually written rules being a part of such a process[1]. The real power of the rule-based approach is in the ability to generate rules from a syntactically preprocessed input and a rule template list. The (supervised) learning is performed on a relatively small set[2] of manually syntactically processed sentences. Manually in this sense means correctly syntactically analyzed

[1] It is possible to add manually designed rules to the set of automatically obtained ones.
[2] Opposed to the statistical techniques, the rule-based learning procedure requires less data. Notes on the success rate and training set size are included in Chapter 3.

sentences. The set of manually analyzed sentences used for the process of learning is usually referred to as the *truth*.

Before one starts with applying the rule-based approach, *templates* of the rules must be postulated (given externally to the system) and a procedure, a function that will decide how well a specific instance of a rule instance (rule) performs, i.e. comparison function between the truth and the actually modified input, known as the *error function* must be known. The learning can be schematized as in Figure 1.2.

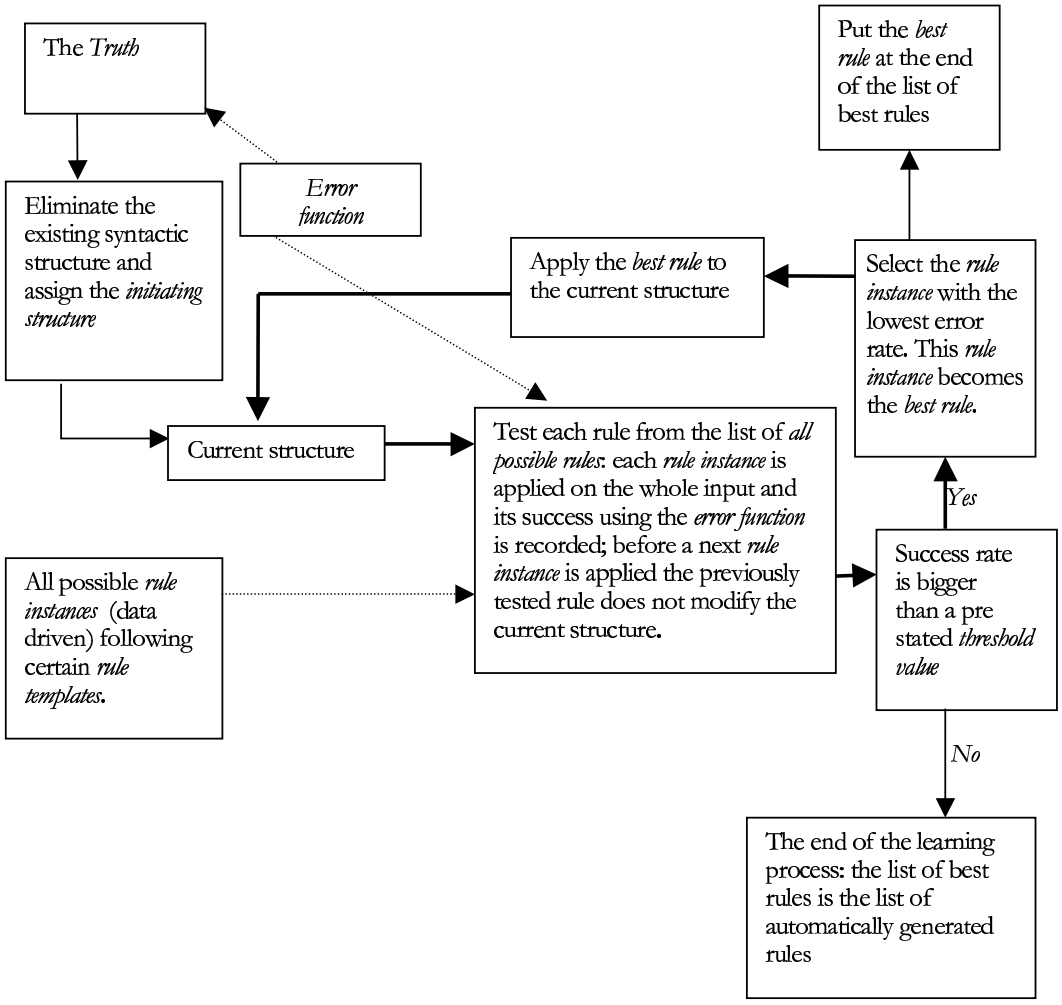


Figure 1.2: RBA - Learning

There are the following connections that have to be preserved between RBA during application (Figure 1.1) and RBA during learning (Figure 1.2):

- The *initial syntactic structure* while learning should be the same as the one while application.
- The error function used while learning is usually, but not necessarily, the function used during evaluation.

- If, as stated in Figure 1.1, the input is enriched with additional information e.g. morphemic, than the same enriched input needs to be present also during learning.
- The list of best rules obtained in the process of learning is the same list of rules used during the process of application, preserving the very same order in which the best rules have been obtained.

Formally, each rule has the following structure:

```
<action> <what> <where>
```

where,

`<action>` is the action to be performed, usually addition, change or deletion of certain elements;

`<what>` determines the element on which the action is performed (for the parsing it is either a constituency boundary or a dependency), and

`<where>` defines of the *triggering environment*, a condition which has to be true in order to perform the action.

For a rule to be performed the data as conditioned in `<where>` (together with the element specified in `<what>`) must be identified and the action specified in `<action>` has to be possible with respect to `<what>` within `<where>`. E.g.

```
DELETE LEFT_PAREN BETWEEN NOUN NOUN
```

would mean: delete a left constituency boundary between two neighboring nouns. In order to perform this rule two neighboring nouns must exist (selection of the environment) and a `LEFT_PAREN` between the two nouns must exist (the first `NOUN` is a father of the second `NOUN`); otherwise the rule will not cause any change to the syntactic tree.

After a rule is applied, technical adjustments of the modified structure need to be performed, the aim of which is to maintain a connected, coherent tree syntactic structure: in a linearized dependency tree description using brackets, deleting a left constituency boundary (left bracket) cannot be left alone without deleting the matching right constituency boundary (right bracket); further, the bracketed string after such modification would not necessarily represent a tree structure anymore, therefore additional maintaining steps need to be performed in order to guarantee that the bracketed string will always represent a tree syntactic structure. The modification steps can be of various type and depend on the specific implementation.

For specific information on basic ideas how RBA is applied to syntactic parsing of English within a constituency based framework, or how the rule-based paradigm is applied to PP attachment, or to POS tagging or to other areas of NLP, one is advised to consider (Brill 1993c).

1.2. First modifications of RBA

Originally introduced and explained only for the constituency grammatical framework, in order to be applied for a dependency framework, as mentioned earlier, the rule-based approach needed to undergo deep changes, which are

mostly visible principally in two of its components: the rule templates and the error function. A closer and more detailed elaboration of the changes will be given for the more valuable second modification of RBA presented in Chapter 3. Here I would elaborate the very first modifications being made, the logic of which is present also in the sequential work.

The following rule templates which operate on the linear description of the sentence and assign only the bare tree syntactic structure (not the syntactic functions) have been proposed:

Rules	**Explanation**
`SWAP LPAREN BETWEEN A B`	Swap A and B if there is a left bracket between them
`SWAP COMMA BETWEEN A B`	Swap A and B if there is a comma between them
`ADD LPAREN BETWEEN A B`	Add left bracket between A and B if there is no left bracket between them
`DEL LPAREN BETWEEN A B`	Delete left bracket between A and B if there is any

Table 1.1: Dependency rules[3]

For clarification and for better understanding of Table 1 and of the bracketing system, the following linear description accompanied with its unambiguous dependency tree structure is shown:

```
A ( B , C ( D , E ) , F ( G ) )
```

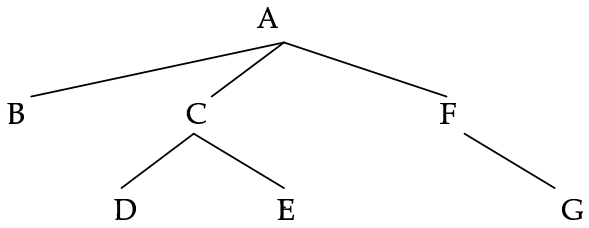


Figure 1.3: An abstract dependency tree

Each left bracket has its "nominated" governor: as A for the left bracket between A and B, or as C for the left bracket between C and D, or as F for G.

Compared to the rules used for obtaining immediate constituents (Brill 1993a), the rules in Table 1 imply the following major differences:

[3] Four other rules were also considered dealing with apposition and coordination: ADD/DEL APOS/COORD BETWEEN A B. These rules added a new node to the tree such that joined two coordinated subtrees or made a relation between two subtrees in apposition. Due to the agreement present in PDT that no extra nodes can be added, coordination and apposition became syntactic attributes, analytical function values, assigned to a certain lexical item representing the relation of coordination or apposition in the tree structure as the conj. "and" can be a coordinating node for Peter and Mary in e.g. "Peter and Mary bought a present".

- They delete brackets, without adding new ones.
- They add new brackets, without deleting some previous ones.
- They change the word order in a sentence.

The following table presents the conditions under which the rule templates are defined.

Rule	**Defined if:**
`SWAP LPAREN BETWEEN A B` `DEL LPAREN BETWEEN A B`	There is only one left bracket between A and B (that is: B is the leftmost son of A)
`SWAP COMMA BETWEEN A B` `ADD LPAREN BETWEEN A B`	There is no left bracket between A and B (that is: A and B have the same governor and B is next to A)

Table 1.2. Rule conditions

If one manipulates with a tree node, then this has an influence on its subtree. The rules containing `SWAP` change the order of words in a sentence. This has a direct influence on the possible triggering environments, which change after each selection of the best rule within the RBA learning process, due to the fact that the surface word order is not preserved[4]. This significantly slows down the learning speed.

An application of each of the rules specified above results in the following modifications presented in Figure 1.5 to Figure 1.8 which exemplify the rule changes on the abstract subtree structure A (B (C , D) , E) from Figure 1.4.

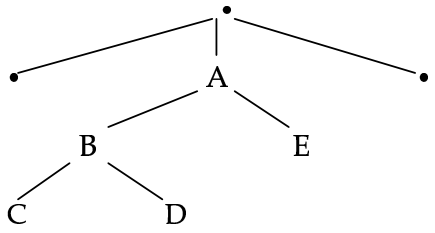


Figure 1.4: Subtree A(B(C,D),E)

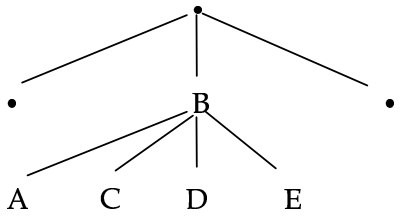


Figure 1.5: Result from SWAP LPAREN BETWEEN A B

[4] Within the approach based on constituency structure the word order is stable. With our framework, possible changes between the surface word order and the order of items in our linear tree representations significantly increase the complexity of the learning process.

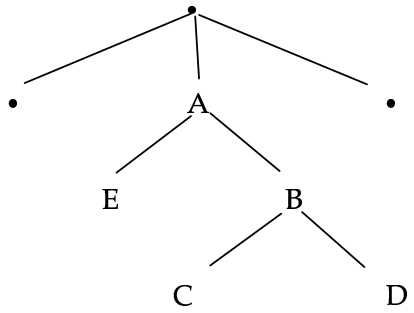


Figure 1.6: Result from SWAP COMMA BETWEEN B E

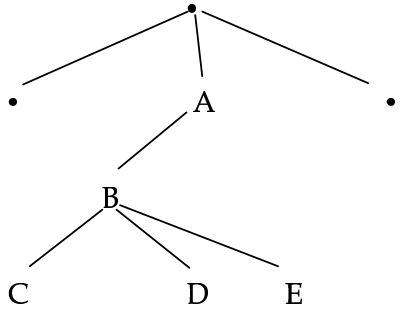


Figure 1.7: Result from ADD LPAREN BETWEEN B E

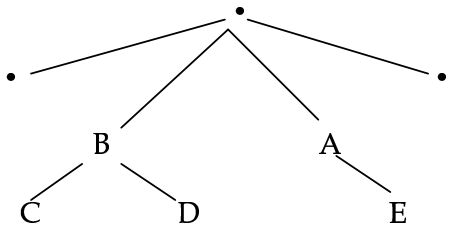


Figure 1.8: Result from DEL LPAREN BETWEEN A B

The error function is completely different from the one used for obtaining the phrase structure that counts the crossing constituents, therefore direct comparisons of the success rates between immediate constituent structure and dependency structure is not possible. A function sensitive to differences of the syntactic structure considered as important was created. It is not advisable to have an error function that would be too sensitive for certain types of changes and unequally insensitive for other types of differences in the tree structure. The error function counts the dependency relations present in the tree with respect to the total number of correct dependencies.

For the tree structure A (B, C(D (E)), F), where the capital Latin letters stand for a tag or a word; there are 5 dependencies as follows:

A(B, A(C, A(F, C(D, D(E.

Let us compare the above analysis to another tree structure of the same sentence:

A (B (C (D, E)), F),

which has the following five dependencies:

A(B, A(F, B(C, C(D, C(E.

The two structures have the following three common dependencies:

A(B, A(F, C(D,

therefore, if one of the structures is 'correct' the success rate is 3/5. If not stated otherwise, this is the way the *success rate* is calculated as an alternative term for *dependency accuracy*.

In order to initiate the process of learning and/or the process of parsing an initial tree structure needs to be assigned to the input sentence to be processed. The *right chain structure* has been chosen, as demonstrated on the following examples, to be the initial tree structure:

(A), for a sentence of length 1, or

A(B), for a sentence of length 2, or

A (B , C), for a sentence of length 3, or

A (B (C), D), for a sentence of length 4, or

A (B (C (D)) , E), for a sentence of length 5, and so on.

RBA is sensitive on different parameters present in the data (as, e.g., length of sentences) or settings of the learning process (as, e.g., initial condition) which indirectly influence the results. The tag set size and its structure is one of the major factors (for Czech, the full tagset size is approx. 3500 tags) that influences the success rate. However, for syntactic tree assignment, the tag set does not need to include all morphemic features (Ribarov 1996). Therefore, after the tag set had been restricted (while still being larger than the tag set of the Penn Treebank), some small size experiments were carried out.

At the beginning (when there was no treebank and there was a need of testing the pure method as such) different very small training sets were created, which contained approx. 25 sentences (the rule-based approach, without further improvements, does not perform significantly better if trained on more than 100 sentences). The average success rate of initial bracketing was estimated to be 42%. Various sets of rules for each training set were obtained consisting of approx. 27 learned rules. The rules were tested on a set of 20 sentences. Each of the set of rules increased the success of bracketing by approx. 8.5%. Those were the very first steps: far from impressing results and astonishing percentages, but still not demotivating ones. The rules were rather simple, and the success rate for the dependency structure is always more pessimistic than the e.g. that concerning crossing constituents one in the immediate constituents structure, and the improvement of 8.5% percent was judged to be relatively not a bad improvement. It was clear that it could be amended in various aspects. Many changes were made in-between, both in the data and in the modifications of the

stated approach. These changes, e.g. caused that the initial bracketing success became approx. 27%. Everything has been done with a primary aim to state the strength and the potential of the automatically generated rules, guided by (secondary but more important) aim of constructing a suitable parser for Czech.

Before I proceed with the analysis oriented mainly to the rule-based approach I would like to devote few lines to the problem specification, the way parsing is understood here, and to make links to the previously works done at the department from where also this work emerges.

1.3. On the word parser

What is a parser? This question can have a very trivial but at the same time rather complex answer. To start with, I will use the "trivial" one stating that a parser is a process of assigning a (labeled or unlabeled) syntactic tree structure to a sentence at the input; the input originally does not exhibit any kind of syntactic information. By a syntactic tree structure (being assigned during the parsing) I denote such syntactic structures as those being present in the treebank in use.

Let IN be an input sentence and SYN_OUT be the same sentence enriched with its syntactic information. Parsing can be defined, e.g. as an algorithm transforming IN into SYN_OUT. Therefore, parsing can be viewed as a mapping between sets of INs (S) and SYN_OUTs (**S**):

parsing: S → **S**.

The desired mapping to be performed is a very sparse one. To support this claim the number of combinatorially possible structures is given in the second column of Table 1.3 (if, as in the case of binary constituents as used in (Brill 1993b)) opposed to linguistically located structures in the Penn Treebank in the third column (the combinatorial space of a dependency tree is even larger). Their ratio is presented in the fourth column. Assuming that a sentence of 20 tokens (including punctuation) is very likely, the ratio of found structures over all possible structures is of a value of 2.4×10^{-8}. An algorithm using no linguistic knowledge will have to test all of those possibilities before selecting the correct parsing. The situation is even worse for a dependency parser (2.6×10^{23}). From this perspective, the algorithm of automatic grammar generation could be specified as an algorithm of effective linguistic knowledge search within an extremely sparse combinatorial space.

i: sentence length	T(i) = (i-1)T(i-2) + T(i-1) possible binary constituent bracketings	Found constituent bracketings in Penn TreeBank	Ratio: the previous 2 columns	Number of trees with i vertices
6	48	25	0.52083333	1296
7	156	24	0.15384615	16807
8	492	40	0.08130081	262144
9	1740	65	0.03735632	4782969
10	6168	70	0.0113489	100000000
20	15566830368	379	$2.4347 \cdot 10^{-8}$	$2.62144 \cdot 10^{23}$

Table 1.3: Possible trees

The most traditional understanding of the parsing would be if *parsing* is such mapping which assigns all possible syntactic structures to the input. This is inherent mainly to hand crafted systems (e.g. manually designed formal grammar parsers). As for the automatic procedures *parsing* is not a mapping but a function, a single structure is assigned to an input sentence. Although statistical parsing algorithms can be programmed to output not only the most probable output but also the less probable ones, thus giving more than a single output (within the rule-based approach it is more difficult to make such modifications), generally speaking statistical and rule-based techniques (and other of similar nature) are not capable of a complete parsing of the input sentence.

I have in mind that parsing primarily denotes an analysis according to the traditional understanding, nevertheless, I will use the term *parsing* (and *parser*) for the automatic procedures for syntactic analysis at any level of the syntactic description. These automatic procedures, as noted above, give mainly a single syntactic analysis of the input stream, although the input might be syntactically ambiguous. In certain parts of this work instead the term parser or parsing, I will also use, e.g. dependency tree finder, building of a dependency tree, and similar.

As implicitly mentioned the structure of the output 'mimics' the structure present in the treebank. The assignment of a parse to a sentence may be viewed in two steps:

(a) assignment of a tree structure to the sentence, and
(b) assigning syntactic attributes to the sentence (dependency) elements.

Three scenarios are possible: first (a) than (b), first (b) than (a), or (a) and (b) together.

I would like to have more freedom for the discrete representation of the syntactic structures being the output of the parser: assigning a syntactic structure to the input does not have to necessarily mean assigning only a tree structure. If a non-tree structure is assigned, it should be transformable to a syntactic tree structure without loss of information with respect to the application such module is a part of.

The syntactic structure to obtain from its informative aspect is a surface syntactic structure (analytical structure)[5] as defined and used in PDT. A *parser* will be the algorithm that does the *parsing*, i.e. assigning an analytical tree (*parse*) to a sentence. The word parser is used only as a matter of naming convenience.

1.4. The theoretical framework and automatic learning

Which information is legitimate to be used for automatic acquisition of the syntactic patterns?

In order to answer this question let us have a look on which is the background one may rely on for handling Czech:

[5] Procedures from automatic transformation of analytical trees to tectogrammatical ones is under development as, e.g., in (Böhmová 2001).

- Linguistic formal description of the language system in the framework of Functional Generative Description (FGD).
- Prague Dependency Treebank with two types of structures: surface - analytical trees and deep - tectogrammatical trees.

The Functional Generative Description (FGP) has its first firm outlines postulated by Petr Sgall at the beginning of the 60's and later published (Sgall, Hajičová, Panevová 1986). FGP is a central theoretical framework for any formal processing of Czech, which has as its main form a system able to generate structural descriptions of all correct Czech sentences.

Immediately after the ideas of FGP were postulated two main streams of research followed: a formulation of the generative component (tectogrammatical description[6]) being the first stream, and an elaboration of the translational (elements between the layers of the theoretical description of the language; in Czech, "překladové složky"). The practical realizations were done on a grammatically simplified but theoretically exhaustive core subset[7] of Czech.

Clearly, this and the work done after, belong to the efforts for mainly formal synthesis of Czech. The analysis of Czech as such is actively done in the 90s and nowadays.

From the point of view of the Prague School the first main step concerns the level(s) of sentence structure, which within the dependency framework display the disambiguated structure of a sentence belonging to a context, marking the dependency relations and positioning the main weight to the verb and its modifications. The mentioned layers of the sentence structure according to the latest considerations are:

1. phonemic,
2. morphonological,
3. morphemic,
4. tectogrammatical.

Between the 3rd and the 4th level an intermediate 'analytical' level has been inserted as a technical device, which has no theoretical status and differs from what in the past theoretical frameworks was assumed to constitute a level of surface syntax. The tectogrammatical level is the deep syntactic level, sometimes referred to the knowledge representation of the sentence.

---

[6] It has still been not clear at that time how dependency based syntax should be treated and whether the description could be fully based on a dependency. Therefore, the description was a combination of constituency and dependency.

[7] (Hajičová, Sgall 1980), (Plátek, J.Sgall, P.Sgall 1984) and (Sgall, Hajičová, Panevová 1986) redesigned the generative component so that it became fully dependency based while V. Petkevič in his PhD thesis has added the missing elements of the description such as, besides others, coordination and apposition (also known as the third dimension in the formal description). Except for (Petkevič 1995) Petkevič's contributions have not been published. Petkevič's description is the most profound formal description of the generative component.

The modern applicational approaches currently under research experiment with the possibility to obtain directly the tectogrammatical structure of the sentence from the morphemic layer at least as successfully as the surface syntactic structure is obtained. The present contribution is not a part of these trends but it neither denies that such direction is possible.

Starting from theory and going towards application, the Prague Dependency Treebank (PDT) was created (and is continuously under refinement). The PDT follows the theoretical framework, which is reflected in its annotation having a three level structure:

1. *Morphemic level* - morphemic annotation
2. *Analytical level* - assigning of syntactic structures with their surface dependency (analytical) functions
3. *Tectogrammatical level* - obtained from the analytical level when the analytical tree structure is transformed into a tectogrammatical one (assuming additions of new nodes and/or deletion of existing nodes) and tectogrammatical functions are assigned.

The analytical level is declared to be a working intermediate level, also known as transition level between the morphemic and towards the tectogrammatical one. The aim of the analytical level is to ease the process of obtaining the tectogrammatical structure.

If not mentioned otherwise, by a syntactic dependency structure of a sentence I refer to its analytical structure, although strictly theoretically the dependency structure of the sentence is constituted by its tectogrammatical structure.

Although the analytical level is well defined (Bémová et al. 1997) the fact that the real aim is to obtain the tectogrammatical level gives one a right not to follow literally the definitions valid for the analytical level, if such could serve the aim in a better way. E.g. the auxiliary nodes (see analytical function's descriptions, e.g. in (Hajič 1998)) which are omitted on the tectogrammatical level need to be recognized but not necessarily included in the syntactic representation.

The information of the morphemic level serves to reach the analytical level, which helps to reach the tectogrammatical level. Let us assume that there are algorithms the outputs of which correspond to the levels as described in the linguistic theory. In this single direction from the surface form towards the meaning of the sentence of the processed language, the input of the 'analytical algorithm' is the output of the morphemic one, the input of the 'tectogrammatical algorithm' would be the output of the analytical one. All of them constitute a chained process, the phonemic (or graphemic) form being at the input of the first of the algorithms in the chain.

The process of chaining can be very strict

(i) the input must be only the output of the (intermediate) level situated immediately lower in the hierarchy,

or the same can be more loosely stated as

(ii) the input can be anything being lower in the formal description.

These views are commonly shared among computational linguists trying to invent the needed algorithms. More often (ii) has the following interpretation:

(iii) the input can be anything that one can find in the treebank at hand.

There is a need of considering these aspects since in the automatic learning procedures the knowledge while learning could differ from the knowledge the procedure should produce, although such approach was not yet taken in consideration but neither excluded. For the process of learning of grammatical relations one may use as a supervisor during the learning procedure the analytical trees and/or even the tectogrammatical trees. Nevertheless, the structure to be assigned during main present-day applications is the analytical tree structure.

(i), (ii) and (iii) are taken to be the legitimate approaches for the process of learning, including (iv) in whichever phase of the analysis;

(iv) The intrinsic knowledge of the language as a whole, the knowledge of the language as a system.

The present algorithms for automatic linguistic knowledge extraction have no effective mechanisms of how to acquire (iv). The knowledge of the language is implemented within the structure of the learning process (the templates of the rules for the rule-based approach, the design of the neural network, the language model within the statistical procedures) and as such needs to be given by humans and results from linguistic studies of the language.

1.5. On analytical and tectogrammatical level

Having fresh the ideas from the previous section, I want to contribute here briefly to a better understanding of the 'technicalities' of the analytical level and in a similar vein its differences from the tectogrammatical one. I will devote myself only to the aspects relevant for the automatic approaches, the minimum one should have in mind before approaching them. Detailed discussion and descriptions are in (Bémová et al. 1997), (Panevová et al. 2000) and other relevant papers as (Hajič et al. 2003)[8].

"Technical" facts relevant for the analytical level:

- Each sentence is represented by a single analytical tree.
- Each word form is a single node in the tree. Punctuation is a separate form, therefore each punctuation mark has its node representative.
- There are no other extra nodes in the tree representation except for one technical node representing and governing the whole sentence. This technical node follows a regular pattern and can be assigned/identified with 100% success rate.
- The dependency relation is coded by the edges, which constitute a governor-dependant relation.

[8] For other works see PDT documentation.

- The analytical functions are assigned not to the edges but to the dependant nodes, thus each word form is a bearer of a certain (its own) analytical function.
- Each node has an analytical function.
- The analytical functions mark: predicate, subject, object, adverbial, complement, attribute, auxiliary verbs, coordination, apposition, reflexive particles, preposition, conjunction, particles, punctuation and other graphical symbols, specific ellipsis handling. Where unclear predefined combinations of pairs of the analytical functions is possible.

Besides the analytical function, each node contains the information inherent to it from the surface form of the sentence and other types of information as e.g. the lemma and the morphemic tag. Since the dependency tree does not preserve the surface word order each node contains its position (order) in the sentence, such that the sentence can always be reconstructed to its original surface form. The order of the words influences the way the dependency structure is represented - it is claimed that the order introduces a second dimension (left to right) ordering to the tree structure (top to bottom).

To illustrate this let us take the following linearized trees (1) `A(B,C)`, and (2) `A(B(C))`. Those can be graphically represented as follows:

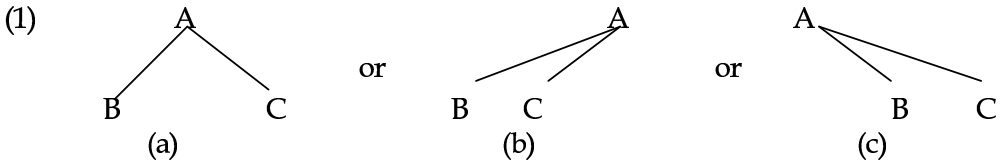


Figure 1.9 (1): A(B,C)

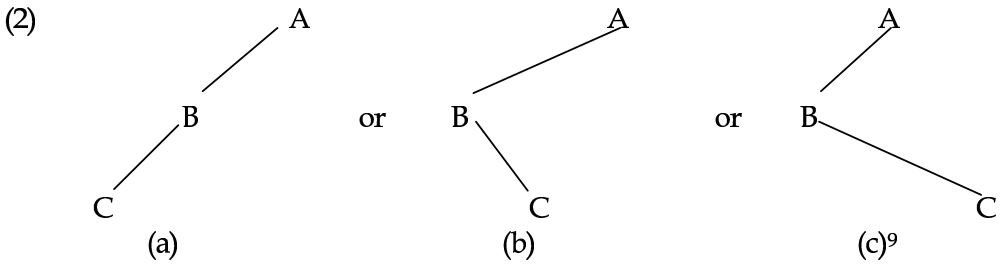


Figure 1.9 (2): A(B(C))

It is the surface word order that makes them distinct. If ord(X) is a function returning the surface order of the form (node) X, here is what is valid for the above trees:

(1)(a) ord(B) < ord(A) < ord (C)
(1)(b) ord(B) < ord(C) < ord (A)
(1)(c) ord(A) < ord(B) < ord (C)
(2)(a) ord(C) < ord(B) < ord (A)

[9] This is known as a non-projective dependency tree, while all of the others are projective: a virtual vertical line below A crosses the B-C dependency.

(2)(b) ord(B) < ord(C) < ord (A)
(2)(c) ord(B) < ord(A) < ord (C)

The linearized form of the tree accompanied by the ord(.) function unambiguously determines the dependency tree.

"Technical" facts relevant for the tectogrammatical level:

- The representation structure is a dependency tree.
- It is obtained by transforming automatically the analytical tree and consequent manual processing.
- Nodes present on the analytical level can be deleted. This concerns especially function words.
- Only an autosemantic word is a node.
- Extra nodes not corresponding to a lexical item from the surface can be added.
- The consideration of the second dimension (left to right) is present and is of special importance.
- New, tectogrammatical, functions are assigned to each node. These functions are of the following type as actor, bearer, patient, and others.

Figure 1.10 presents an example of an analytical tree, and Figure 1.11 presents the tectogrammatical tree of the same sentence "České radiokomunikace musí v tomto roce rychle splatit dluh televizním divákům." (*The Czech radiocommunications have_to in this year fastly repay debt to_television viewers.*).

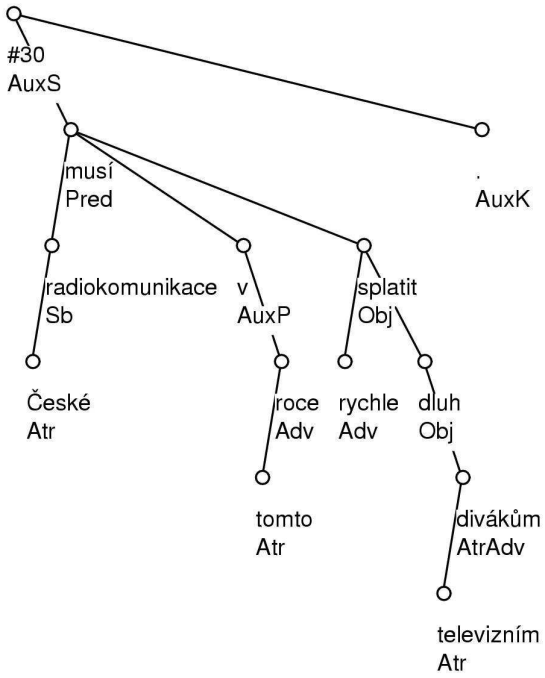


Figure 1.10: An example of an analytical tree

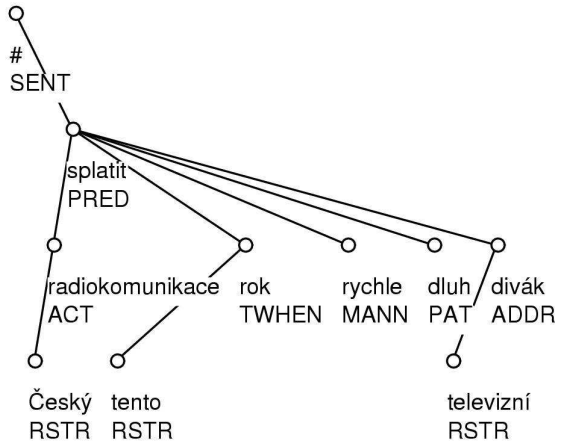


Figure 1.11: An example of a tectogrammatical tree

Currently, the sentences from PDT can be searched using:

- the NetGraph tool (Mírovský NetGraph), and
- the TrEd tool (Pajas TrEd).

1.6. Early parsers and processing of Czech

Although not directly related to automatic syntactic knowledge extraction, there has been a significant piece of work already done on the field of formal processing of Czech. It is my intention to give here a brief glimpse of the past works starting from the late 50's. Unfortunately these past works cannot be directly used or re-used, but the experience and the theory verifications they provided constructs a chord to respect between the obtained linguistic results and new tendencies for automatic learning as this work is. The work presented in the following chapters does not have any direct predecessors for Czech.

Most visible of all, RUSLAN (containing in its name, besides others, RUSsian LANguage) and TIBAQ, (Theory and Inference Based Answering of Questions) are predecessors of any more complex automatic processing of Czech including proper syntactic analysis. They share a common structure as presented in Figure 1.12.

Both TIBAQ and RUSLAN, as described in (Hajičová 1995), (Oliva 1989), have the inflected words split to stems and endings, the dictionary of endings, the information from stems and endings conjoined and the syntactic-semantic analysis in common, while the other modules have the same name but have different content. The syntactic-semantic analysis has been constructed with the help of a dictionary (as in Figure 1.12) The dictionary consists of relevant syntactic and needed semantic features such that the grammar basically can consist only of two types of rules: most general rules for Czech and Russian syntax like rules concerning subject-verb agreement, and highly schematized rules processing the relevant lexical information taken from the dictionaries like

rules for filling the slots of case frames or rules carrying out certain changes of word order if such changes are required etc.

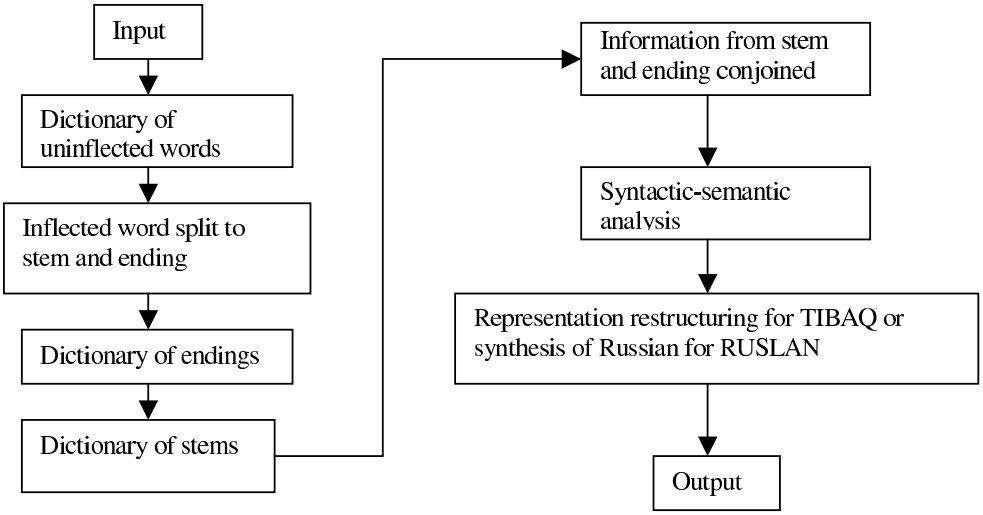


Figure 1.12: Early structure for processing (parsing) of Czech

The implementation of the mentioned systems was done by Q systems. Designed by A. Colmerauer, Canada, the Q systems were a very attractive tool besides others, for their possibility to implement dependency-like structures (any description for an implementation of a language formalism at that time were based on constituencies). They were also easy enough, opposed to the then present low level languages and Fortran, such that a trained linguist could program them himself. However, those who have had direct experience with programming the Q systems also mention the lack of transparency in the design of a complete system: there were so many Q-systems to maintain simultaneously that their control had been almost not possible. More detailed description can be found in (Oliva 1989) with their application for the process of parsing.

Undoubtedly, the analysis and synthesis present in TIBAQ and RUSLAN[10], are of a high value and importance. On the part of the analysis (which used a very limited lexicon and syntax), the input sentence (one may assume that the input sentence has been morphologically analyzed and tagged) is processed using the mentioned Q-systems producing a tectogrammatical-like[11] structure at the output. A file of tectogrammatical representations of sentences formed the knowledge base of the system (TIBAQ is an answering system: the user poses a question, and the system searches for a relevant answer from the knowledge base). The question at the input was also processed and its tectogrammatical-like

[10] Summarization of RUSLAN can be found in (Kuboň 2001).

[11] Not tectogrammatical but tectogrammatical-like, because of the implementary modifications done to ease the processing and at the same time because of the differences with what is now-a-days assumed to be tectogrammatical after modifications were implemented in the past decades.

structure was obtained, after which comparisons with inference based procedures were invoked in order to construct the answer. Thus, searching for facts not directly/explicitly contained in the knowledge base was made possible, e.g. "everybody can" implies "A can", where A is an instance (a member of everybody), or "the device A can do" implies "A does".

The beginnings of syntactic processing and analysis of Czech does not start with the mentioned products, although they are its first more known and complex instances. Before that, in 1959 Sgall with his colleagues comes with the first attempt for an experiment with translation from English into Czech. Few years after that another attempt for the English to Czech translation was carried over by Kirschner (Kirschner 1987, 1988). After that a synthesis of the Czech sentence was done.

All of those algorithms were implemented, as the time permitted, on small sets of data just enough to show/test the ability of the machine to verify the formal description.

Although it would be very difficult to imagine, from a perspective of the present much higher computational power, how an analysis could have been efficiently performed, the analysis of Czech has not been totally absent in the past. Mainly in the beginning of the 60's algorithms (theoretically described) were done for morphological analysis (and synthesis), various algorithms for syntactic analysis of Czech were designed, such that partial syntactic phenomena of Czech were exhaustively studied and those partial algorithms were postulated (unfortunately they were not joined into a single analysis).

Algorithms for searching and information retrieval were done and also implemented, known under the names of MOZAIKA (na MOrfologii Založené Automatické Indexování Koherentními Agregáty) and AZIMUT (Automatické Zpracování Informací Metodou Úplného Textu). Without a use of a dictionary but based on ending segments of the word forms MOZAIKA was able to search successfully for relevant keywords in the Czech.

AZIMUT has been a full text searching engine for the highly inflective Czech language (Králíková, Panevová 1990). Its main power is in the module "Generator" a substitution of a lemmatizer, a module that was able to generate (for a given input) possible relevant word forms under which the input form could occur in the text. The user had also the possibility to search for more than one term joining them with basic logical operators, and controlling the presence of their occurrences within a sentence, paragraph, text.

For all of the mentioned works, a qualitative analysis of the results is presented but without evaluation of the type one is used to receive nowadays. Therefore it is not possible to say to which extent (in terms of percentages) does this knowledge and previously done work help our current analysis. Neither it is possible to retest the algorithms since that would require reprogramming them. Reprogramming would not mean only recording them, but also redesigning the algorithms (they very often include conditions based on e.g. valency, which is a value that we should obtain and not a value we have).

Nevertheless the past work is extremely motivating and valuable for an analysis one performs now-a-days. In fact, analysis on the morphemic level (morphological analysis and lemmatizing) algorithmized and completed by Hajič (Hajič 1999), benefits from the past descriptions of the morphemic phenomena of Czech (Weisheitelová, Králíková, Sgall 1982), the meta rules used for Kuboň's grammar checker (Kuboň 2001) benefit from the syntactic descriptions of Czech also done in the past. The significant contribution of Lopatková (Straňáková 2001) for prepositional group attachment resolution in Czech is a continuation of the theoretical research trends including as well significant analytical value.

Besides manually coding the linguistic information the richness of language data and the development of the computational power allow for methods of automatic learning, mainly statistical or rule-based; naturally this was not possible in the past. Questions arise how to join former and present efforts.

1.7. Current trials on parsing of Czech

I will mention here only several representatives of two main trends of parsing of Czech:

- the trend closer to the traditional parsing, i.e. generation of a set of syntactic structures relevant to a given input sentence
- the trend of automatic, mainly statistical, assignment of a single, contextually correct syntactic structure for an input sentence.

The first one would like to obtain the highest (possibly 100%) recall and maintain a high recall level while restricting the set of thousands of syntactic structures for each sentence to a small core of true ones. As new representatives of such trend I would select (Horák 2000) and (Žáčková 2002). (Horák 2000) presents an impressive syntactic analysis tool, a chart parsing based tool with additional contextual elements and grammatical agreement tests. It operates with a grammar of the Czech language with a 90% coverage on the Prague Dependency Treebank. A later work of (Žáčková 2002) deals with partial analysis of Czech sentences in a Prolog environment within which she implements rules of grammatical constraints.

The second trend is the one where the authors' aim is to find the best syntactic structure for an input sentence. The aim is to have such best structure that will have the highest possible precision measured on the Prague Dependency Treebank. These approaches are not originally designed to output more than a single sentence structure. The most famous of all is the work of M. Collins (Collins 2002) based on stochastic grammars and later on followed by the successful experiment of E. Charniak (Charniak 2001). In the Czech environment it is the work of D. Zeman who works with dependency probabilistic model (Zeman 1998) and his later work on the same model, which is not published yet and includes significant improvements compared to the state in 1998. The work presented in this thesis belongs as well to this trend.

I would not like to omit also the work (Kuboň 2001) on a grammar checker of Czech which is closer to the syntactic analysis tools, but based on text mistakes by which in its analysis 'phase' grammar rules are triggered.

As a part of a future research it would be interesting to try to combine the two trends, a combination where the automatic parsing (the second trend) would try to help the syntactic analysis for a more successful restriction of the generated syntactic trees (the first phase).

1.8. Breaking the problem

Revealing the analytical structure of a sentence is a complex process. With current algorithms for analytical structure determination, the success rate rises if the process is viewed as a system with various subparts as in Figure 1.13. In this section I would like to describe a situation which was in my mind while creating the parser presented in this work and at the same time a situation in which the parsers can be placed. The parser would be the main module of such a system. The system has the following input and output:

The input: A sentence or a word group acting as a sentence. The sentence can be associated with its morphemic information (see later on modifications of this information and selection of morphologic information relevant for syntactic parsing).

The output: analytical syntactic structure of the input sentence, or the input sentence accompanied by syntactic information such that it can be easily transformed into an analytical tree structure of the input sentence, or into a partial analytical tree that provides at least as much information as needed for the next step of processing.

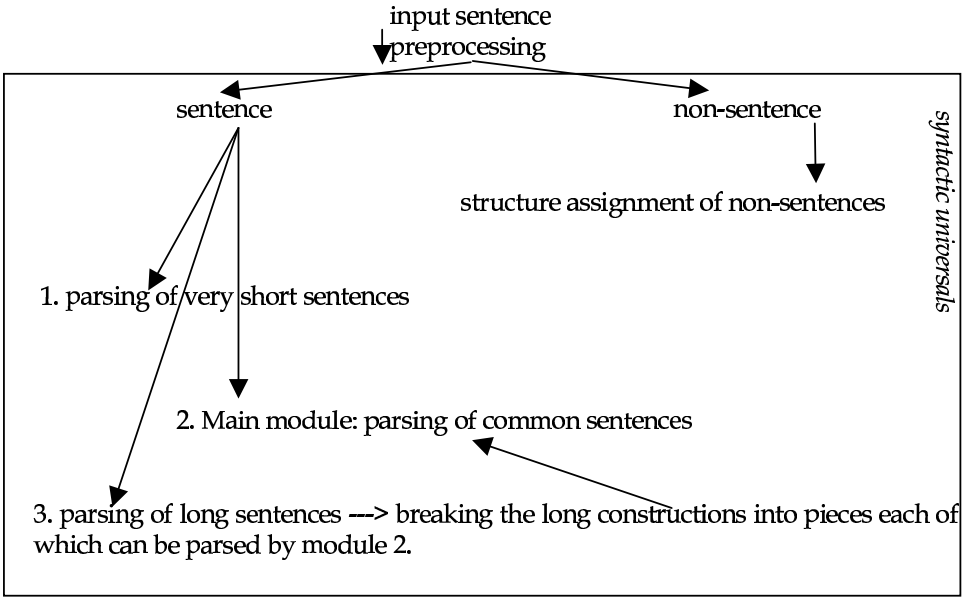


Figure 1.13: A model outline

As stated earlier, a proper output should give all possible parses of the input sentence. Unfortunately, this aim is not possible to fulfill with the methods here in use, although different parses of the sentence are not a priori excluded.

The process of preprocessing could identify any kind of regular and fixed information. Its aim is to localize the regular parts of the text and the regular parts of the sentences, as:

- identification of tables, lists, sport event results (taken to be non-sentences)
- localization of very short and very long sentences (sentences with a large number of verbs or rich punctuation)
- identification of compound verbs (verbal analytical forms)
- identification of prepositions and prepositional groups
- certain coordinations
- certain parenthetical parts
- certain idioms and phrases.

This list can be modified and depends on the language being processed, in this case, the Czech language. The criterion of what rule can be a part of the preprocessing module could be given by the ability of designing a rule such that it will not reduce the recall of the system. The need to separate the long sentences from the shorter ones is a result of unambiguous experiments, which show that the longer the sentence is the less successful is the parser. Breaking the long sentences into smaller sentential segments reduces the combinatorial space of possible structures to be searched. For its possible positive effect, besides the significant influence of distance as described in Chapter 4 see also (Holan 2003), (Ribarov 2000b).

Having in mind, that PDT consists of various texts most of which are newspaper texts included in PDT without any editorial work or selections, one must expect a variety of non-sentences in the input. These non-sentences, to mention some of them tables, lists, sport or chess results, are not our main objects of interest. Although being assigned an analytical structure, such a segment is dependent on the spelling norms of the language and exhibits firm regularities. Therefore, it is believed that if they are recognized correctly, specialized modules based on regular expressions could process them.

During the whole process, verification of certain invariant elements could be performed. As for Czech these elements are to be searched in:

- analytical forms (usually verbal; it is also a part of the preprocessing),
- check on morphological agreement,
- list of atypical syntactic constructions based on the manual for analytical annotation, where such structures are exemplified, taken as a list of structured exceptions,
- results of a study of the discrete nature of the tree structures present in the treebank; non linguistic information that refers to the tree structures and their structural characteristics.

These and possibly other 'syntactic universals' could be applied continuously throughout the parsing process or as a post processing (correction) unit. The syntactic universals can always be also a part of the preprocessing unit. The way how the syntactic universals are applied influences the success rate of the system. An interesting and useful work on combination of regular expressions and statistical parsing is done as in (Zeman 2001).

Speaking of parsing very short sentences I have in mind parsing sentences without verbs. In such sentences one may expect incomplete grammatical constructions having originally other than basic narrative, exclamatory or interrogative character in the text. Usually headers or other titles have these characteristics. It is justifiable to assume that such sentences can be successfully localized, which would allow, if needed (e.g. the rule-based method specially trained on such sentences could parse them in a satisfactory manner) to process them separately.

The main (parsing) module is independent on the type of the parser being used, but an automatically trained parser being either a rule-based one or a statistical one is expected. Any combinations of the both are not excluded.

2. Some Aspects of PDT

The aim of this chapter is to present some characteristics of the object of study, the Prague Dependency Treebank (PDT). The presented characteristics are with respect to the performed experiments in the later chapters and with respect to automatic parsing algorithms in general. The characteristics are presented as frequency counts on PDT, such that help us sharpen our intuition and judgements towards algorithm performance.

2.1. Using PDT

PDT is the largest collection of syntactically annotated Czech data and, in general, one of the largest such collections in the world. PDT has a unique depth of analysis at three levels: morphemic, surface syntactic (analytical), and deep syntactic (tectogrammatical). A fourth level is also envisioned - a logical propositional one. For more detailed information on PDT, see (PDT 1.0).

The level of our interest will be the analytical one. It consists of almost 100,000 sentences divided in three groups: train set sentences, development set sentences and evaluation set sentences. The files can be accessed in two different formats: an internal feature structure format `*fs`, and a SGML based format with its own DTD. Using ready tools as NetGraph (Mírovský NetGraph) or TrEd (Pajas TrEd) one may easily access any kind of information from the annotated material independently on the input format. Moreover, there is a conversion tool between the two mentioned formats available (see PDT 1.0). For doing any kind of a heavier operation on the treebank data, the bTrEd (batch variant of TrEd) and nTrEd (network variant) are irreplaceable tools. TrEd allows a direct access to the trees and the node info using a Perl code and macro sequences parsed and processed by TrEd. Traversing a tree in PDT is as simple as including something as the following code, which is passed as an argument to btred.

```
sub autostart {
        # code before
        do {
           while ($this) {
                # code in

                # In this part we may access any kind of
                # information of the node, e.g.
                     # $tag = $this->{tag}
                     # $lemma = $this->{lemma}
                     # $form = $this->{form}
                     # $order = $this->{ord}, etc.,
                     # or access, e.g.,its parent by
                     # $this->{parent}

           $this=$this->following;
           }
        } while NextTree();
        # code after
}
```

2.1.1. Node values of interest

Tags of special interest for this work are the following: *ID1* for sentence identification, *form* for a word form and *lemma* for its lemma, *ord* for identification of the surface word order, *tag* for the correct (manual) morphemic tag, *afun* for the analytical function assigned to the word form.

The morphemic tag is a positional one with 15 positions in the following order: (1) Part of Speech, (2) Detailed Part of Speech, (3) Gender, (4) Number, (5) Case, (6) Possessor's Gender, (7) Possessor's Number, (8) Person, (9) Tense, (10) Degree of comparison, (11) Negation, (12) Voice, (13) unused, (14) unused, (15) special usage.

The PDT tree nodes contain also tags which are the output of: a morphological analysis of Czech, output of two different best taggers for Czech. Specially the latter are used when chaining the, e.g. syntactic module, to previous morphemic ones.

There is one artificially added node, the root of the sentence, which is assigned an ord of 0 and tag of Z#----------.

The presented statistics are calculated over the whole set of 1583 files of analytical for general training subset of PDT.

2.2. Basic facts

Major part of automatic learning procedures are directly dependent on sentence length, usually respecting a law that the longer the sentence is the less successful the automatic procedure is. The sentence length with respect to the number of nodes is presented in Figure 2.1.

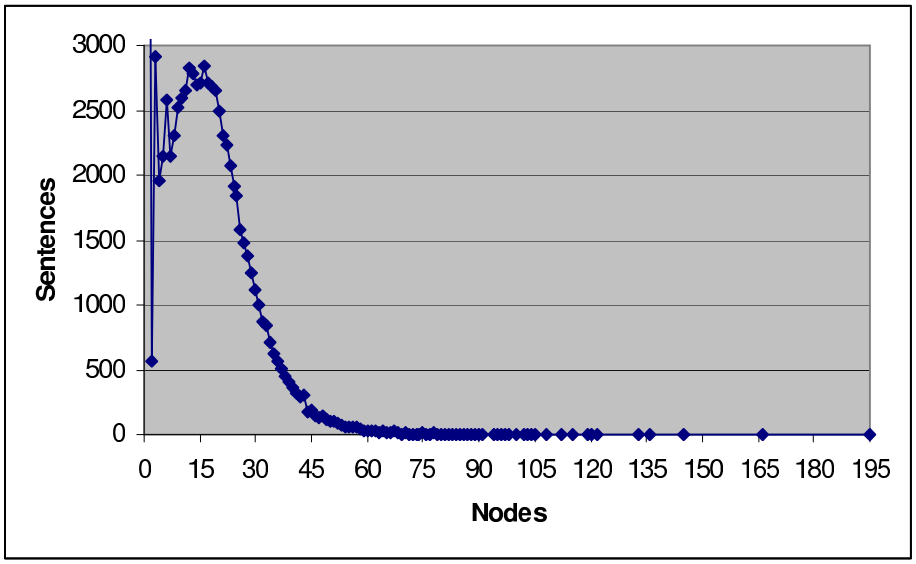


Figure 2.1: Sentences and their length

With respect to the sentence frequencies, the average sentence length is 16 nodes, i.e., 15 words (the last, 16th, node is punctuation). If the one node sentences are excluded the average length is 18 nodes. Even further, if we exclude also two node sentences (which are parsed by simply connecting the two nodes, one of which is the artificially added root node) the average sentence length remains 18.

It is not generally true that one-verb sentences are short sentences. It is surprising to see that even a one-verb sentence can have a length of even 60 or more nodes. Nevertheless, although very rich on syntactic elements, one-verb sentences are 'simpler' than those with more verbs. On other hand, sentences without verbs exhibit other characteristics. A parser could be trained separately on these sentence groups. Performance on one-verb sentences is more successful, but within more sophisticated approaches this difference is usually not a big one. Table 2.1 presents number of sentences with the corresponding number of verbs.

# Verbs	# Sentences
0	19307
1	20722
2	17218
3	11059
4	6543
5	3418
6	1714
7	842
8	406
9	181
10	101
11	52
12	23
13	13
14	8
15	3
16	1
20	1
21	1
23	1

Table 2.1: Number of verbs per sentence

The success of the automatic procedures varies also with the cardinality of the tagging set (Hladká, Ribarov 1998). With respect to the past experiments and with respect with those presented later in the text two most common reductions of the tagging set is either by taking only the first two positions of the morphemic tag or its first five positions. Evidence on different dependencies present in PDT for two variants of tagset reductions (the top 30 dependencies) is given in Table 2.2. The reductions are straightforward: (m5) the first five positions of the morphemic tag, (m2) the first two positions of the morphemic tag.

Father	Son	Frequency	Father	Son	Frequency
Z#---	Z:---	73616	NN	AA	118969
Z#---	VB-S-	18710	NN	NN	95806
J,---	Z:---	17347	RR	NN	92695
J^---	Z:---	17327	Z#	Z:	73616
RR--6	NNFS6	12185	VB	NN	49527
VB-S-	Z:---	11627	Vp	NN	38418
Z#---	J^---	11474	J^	NN	36209
NNFS2	AAFS2	10680	NN	RR	33560
RR--6	NNIS6	10520	VB	RR	26319
J^---	VB-S-	9632	Z#	VB	25321
Z:---	Z:---	9533	Vp	RR	24167
VB-S-	Db---	9188	J,	Z:	18081
NNMS1	NNMS1	8942	Z#	Vp	17809
VB-S-	NNFS1	8843	Z:	NN	17744
NNFS1	AAFS1	8611	J^	Z:	17327
Z#---	VpYS-	7893	NN	Z:	17297
NNIS2	AAIS2	7348	VB	Z:	17081
RR--6	NNNS6	6952	J^	VB	13889
VB-S-	RR--6	6931	VB	Db	13418
C=---	Z:---	6658	NN	J^	12978
Z#---	VB-P-	6611	Vf	NN	11662
VB-S-	Vf---	6400	Z#	J^	11474
NNFS4	AAFS4	6394	Vp	Z:	11419
VB-S-	NNIS1	6305	Vp	Db	10355
J,---	VB-S-	6162	J^	Vp	10044
NNIS1	AAIS1	6107	NN	C=	9972
VpYS-	NNMS1	5759	Z:	Z:	9533
VB-S-	P7-X4	5607	Z#	NN	9163
NNIS4	AAIS4	5589	VB	P7	9114
RR--4	NNFS4	5548	VB	Vf	8682

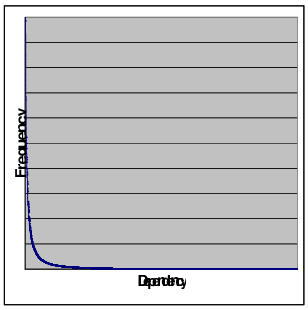


Table 2.2: Dependency frequencies

When only the first five tag positions are considered there are 15,820 different dependencies, or 1,212 for the case when the first two position of the tag are considered. For the first case, almost 2/3 of them have a frequency lower than 6.

Neither bigrams nor trigrams are sufficient to capture the dependencies. Although the longer the dependency is the less probable it is, 76% of all dependencies are longer than 3. A length (distance) of a dependency is the absolute difference of surface order (function ord) of the dependency nodes. The dependency lengths are presented in the figure of Table 2.3, the 15 most frequent of which are presented in Table 2.3. The longest dependency in PDT is 195.

The average dependency length is 11 calculated over the total number of 22,142,414 dependencies found in 81,614 sentences.

Dependency Length	Frequency
1	1992077
2	1713466
3	1562417
4	1449851
5	1347864
6	1251858
7	1162216
8	1074850
9	991961
10	914112
11	839241
12	768198
13	700602
14	639001
15	580709

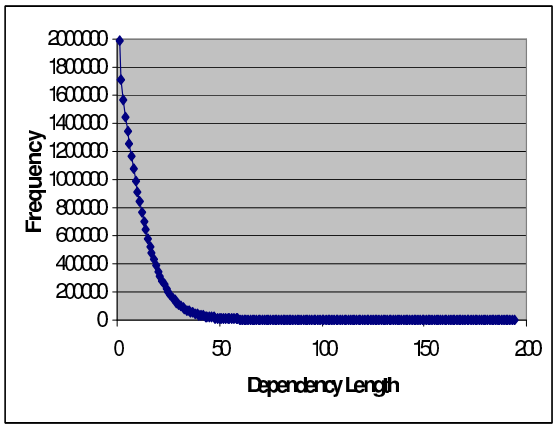


Table 2.3: Dependency length frequency

Unfortunately for initiating studies on automatic training procedures, PDT contains a large number of 'inadequate' sentences, to mention two of them: non-neglectable amount of empty sentences (one node sentences), i.e. sentences with only one node (therefore the success of parsing is always 100%), or opposite to that a significant amount of 'non-sentences', as various enumeration lists or lists of other kind, various table-like data coded as a tree, where the parsing rules are more technical agreements on their tree description than linguistic/syntactic relevant rules. It should be noted that the latter introduces noise that influences a learning process.

There are also many linguistic cases in syntactic analysis (assignment of analytical trees), e.g. those for which the two tree dimensions are not sufficient, as in the case, e.g., with coordination. Hence, the necessity to represent all emerging cases requires employment of various compromises between linguistic justification and technical tree representation.

Besides the difficulties of the language itself, such characteristics do not make our life easier. Therefore, whenever possible, technicalities should be identified and localized.

For additional experiments on dependency frequencies see (Holan 2003).

2.3. Saturation of PDT

Figure 2.2 and Figure 2.3 present the cumulative frequency of newly occurring dependencies over PDT, which exhibits, or should exhibit, a saturation effect. The saturation effect is visible for the cumulative frequencies of tags taken as the first two (m2) morphemic tag positions (Figure 2.3) and it also starts occurring for the cumulative frequencies of tags taken as the first five (m5) morphemic tag positions (Figure 2.2). The more saturated the treebank is, the better it represents the statistical aspects of the diversity and completeness of syntactic information. PDT could be placed (with respect to the full morphemic tag) in a region between

non-saturated and saturated phase. After approx. 80,000 processed sentences for (m2) a new dependency occurs in every approx. 300th sentence, while for (m5) a new dependency occurs in every approx. 10th sentence, which is an onset of a more stable saturation region. On one hand one could always wish a bigger PDT, on the other hand PDT is almost big enough. Following such trend, saturating it would require additional approx. 30,000 sentences or a lower set with a selection of such sentences that will cover missing grammatical constructions.

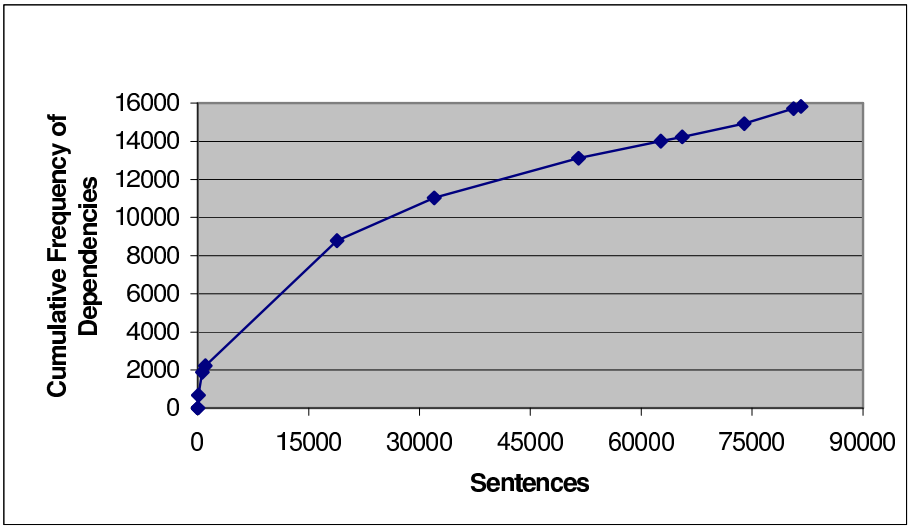


Figure 2.2: Cumulative frequency of (m5) dependencies

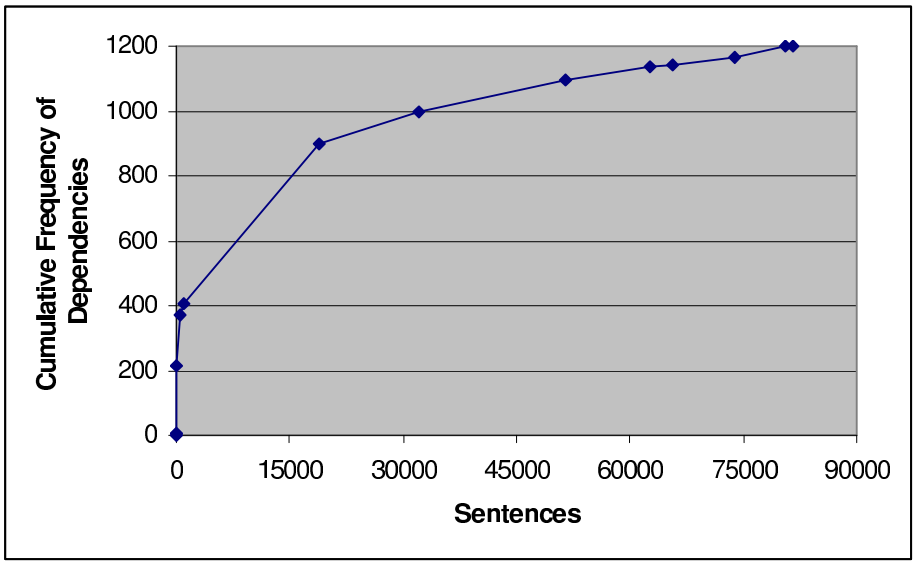


Figure 2.3: Cumulative frequency of (m2) dependencies

3. Further Modifications of the Rule-Based Approach for Czech

After the first attempt to apply the rule-based approach for a dependency based syntactic analysis of Czech (Chapter 1, Sections 1.1 and 1.2) several other steps and modifications were made (Ribarov 2002). Most important of all, the modified RBA was tested on real corpus data, as soon as those were available. Increase of the success rate was observed, higher than the one in the early experiments described in Chapter 1. Although higher, the success rate has not been high enough for any sensible automatic application. To reach the higher (unfortunately still relatively low) success rate, several other modifications were made: the set of rules was redone and enriched with new rules, other error rate function during the phase of learning was considered, various morphemic tag sets were tested. This chapter summarizes those efforts and its content has more an academical value than an applicational one. This chapter contains relevant information for applying the rule-based approach for parsing of Czech. Since the tagset size is relevant for the success rate of RBA, as also demonstrated here, experiments with different tagset sizes (Hladká, Ribarov 1998) are included. At the end of this chapter experiment for tagging a surface sentence form by analytical functions is introduced.

3.1. The input

In all of the experiments, the input text is from PDT, and the rules operate on the morphemic representation of the input text. Thus, the sentence (PDT file c110.fs, sentence number 29):

Těm bude umožněna krátkodobá stáž v zahraničí nebo studium v ČR.

(Them will_be allowed short-term stay in abroad or a_study in CZ.)

which has the following morphemic categories

Těm/PDXP3	bude/VB-S---3F-AA
umožněna/VsQW---XX-AP	krátkodobá/AAFS1----1A
stáž/NNFS1-----A	v/RR--6
zahraničí/NNNS6-----A	nebo/J^
studium/NNNS1-----A	v/RR--6
ČR/NNFXX-----A---8	./Z:

becomes the following input string (the rules do not include the lexical items and take the sentence to parse to be the input stream of morph. tags)

PDXP3 VB-S---3F-AA VsQW---XX-AP AAFS1----1A NNFS1-----A RR--6 NNNS6-----A J^ NNNS1-----A RR--6 NNFXX-----A---8 Z:

The dependency tree is represented in its linear form.

In the earlier stages when no disambiguated morphemic data large enough were available, but one had only the output of a complete morphological analysis

of the input words, there was a need of accustoming the algorithms to that situation. It needed to be stated where it was possible to apply the readings/interpretation of the rules such that the non-disambiguated morphemic tags could be used.

Rules	Explanation
SWAP LPAREN BETWEEN A B SWAP COMMA BETWEEN A B ADD LPAREN BETWEEN A B DEL LPAREN BETWEEN A B	All of the rules are read in the same way as before (Table 1.1) assuming that A and B are not disambiguated tags but members of a set of possible morphemic tags.

Table 3.1: First set of modified rules

The present computational power and the combinatorial character of the learning process does not allow a practical use of a big tag sets and does not allow more than one tag per word. Therefore the requirement is to have a disambiguated morphology and to operate on the disambiguated tags[12], at least for the process of learning (while during application one could assume an input with multiple tags and interpret the rules as in Table 3.1).

The *initial tree structure* is the so-called *right chain tree structure*, which linear description is defined as follows.

> Let A_1, A_2, ..., A_n , A_Z be the sentence surface form and # represent the artificially added sentence root, as in PDT;
> do: T(0) = # (T(1), A_Z) ; T (*i*) = A_i (T(*i*+1)) , for *i*=1...*n*-1; T(*n*) = A_n.
>
> Hence, the result would be
>
> # (A_1 (A_2 (... A_{n-1} (A_n) . . .)) , A_Z)

which graphically corresponds to the tree in Figure 3.1.

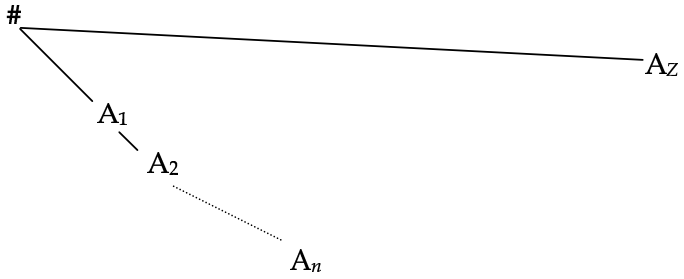


Figure 3.1: A right chain initial structure

3.2. Using modified tagsets

Assuming that the input sentence to parse consists only of morphemic tags, the type and the size of the tagset has a direct influence on the success rate of the parsing procedure. The motivation for this study can be found in the following:

[12] The first results of the learning process were obtained by selecting the first morphemic tag from the list of tags to be the "correct" tag.

- The success of tagging of Czech is influenced by the cardinality of the tagset. As discussed in (Hladká, Ribarov 1998) and also supported by former study by (Elworthy 1995) (where experiments concerning changing tagsets are presented for three different languages (English, French, Swedish) are presented) the tagset should be chosen according to the requirements of a given application. The aim is to join a morphemic tagger and a parser in order to obtain a fully automatic procedure. Therefore the output of the tagger should, in our case, suit best the needs of the parser.
- In order to join a tagger and a parser one also needs to have a tagger with the highest possible success rate (the parser should then be trained on the output of the tagger).

Practically this is realized by examining the performance of the parser on tagsets of different sizes. At the same time the success rate of the tagger itself for the various tagsets should be observed, such that the most successful one is selected.

Three different POS tagsets (POS TAG$_{1171}$, POS TAG$_{206}$, POS TAG$_{34}$), statistical approach to tag text and the tagging accuracy of each statistical tagging with three different tagsets have been developed and tested. Cases of incorrect tag assignment to the words in the input sentence (*Zkrocení zlé ženy mělo úspech* [lit. *Taming of the shrew had success*]) are in boldface in the tagged input sentence:

- for POS TAG$_{1171}$: Zkrocení/**ANS11A** zlé/AFS21A ženy/NFS2 mělo/V3SAMONA úspech/**NIS1**
- for POS TAG$_{206}$: Zkrocení/NS zlé/AFS21A ženy/**NP** mělo/V3SAMA úspech/NS
- for POS TAG$_{34}$: Zkrocení/NOUN zlé/ADJ ženy/NOUN mělo/VERB_PAP úspech/NOUN

The results are presented in Table 3.2, where method BMM is Bi-gram Markov Model, RB is Rule-Base tagger, MMFS - Markov Model realized by finite-state automata.

Method	**BMM**	**RB**	**BMM**	**RB**	**MMFS**	**MMFS**	**MMFS**
POS tagset size	1 171	1 171	206	206	47	43	34
Training data size	600K	38K	600K	38K	15K	15K	15K
Tagger accuracy	81.5%	79.8%	90.1%	87.2%	91.7%	93.0%	96.2%

Table 3.2: Tagset size and success rates for tagging[13]

For a user, let us say a linguist, whose aim is to specify morphemic categories, we need to perform tagging with a full tagset, more specifically, with carefully selected and detailed morphemic classification according to the traditional grammar. The result could be viewed as a process which „added" information to the text, or as a way of classification (clustering) of the word mass. Different

[13] Table 3.2 is taken from (Hladká, Ribarov 1998).

tagset sizes result in different classification of the word forms. The members of each cluster are thus given the same tag.

Previous tagset reductions have been connected with specific mutual dependence between each two of the tagsets. The tagsets mappings have to preserve the patterns, the (ir)regularities of the language.

If the training corpus is annotated with the POS TAG1171 we can observe results of the rule-based approach given in Table 3.3. The rules presented in Tables 3.3 and 3.4 are from the second RBA modification to be presented in Section 3.4 of this chapter.

Id	Description of the Rule	Success (%)
1	Swap the dependency between ZIP and NFS4A	33.67
2	Swap the dependency between ANS51A and NFP5A	34.53
3	Swap the dependency between PDFS2 and NFS6A	35.14
4	Swap the dependency between PQFIP1 and VPS3A	35.74
5	Swap the dependency between RV7 and VPS3A	36.27
6	Swap the dependency between ANS51A and NIS4A	36.81
....	....	....
7	Swap the dependency between ANP71A and NFP7A	44.56
8	Swap the dependency between PQFMP1 and VPP3N	44.76
9	Swap the dependency between ANS53A and NFS6A	44.96
10	Swap the dependency between ANS61A and NNS6A	45.16
....	....	....
11	Swap the dependency between AFS71A and NMS6A	47.18
12	Swap the dependency between ANS61A and NOMORPH	47.38

Table 3.3: Learned rules on a large tagset

The situation changes rather dramatically if we train on the reduced tagset POS TAG34 as shown in Table 3.4.

Id	Description of the Rule	Success (%)
1	Swap the dependency between ADJ and NOUN	44.38
2	Swap the dependency between CM and CONJ	46.00
3	Swap the dependency between PSE and VERB_PRI	47.48
4	Swap the dependency between ADV and VERB_PRI	48.74
5	Swap the dependency between PROP and VERB_PRI	49.58
6	Swap the dependency between ADJ and NOUN	50.42
7	Swap the dependency between CM ADJ	51.29
8	Delete the dependency between ADJ and CM	52.22
....	....	....
9	Add a dependency between PUNCT and PROP	64.84
10	Delete the dependency between PRON_INS and PREP	65.02

Table 3.4[14]**:** Learned rules on a small tagset

[14] The rules in Table 3.3 and Table 3.4 are not the complete sets of rules.

Not all of the rules result in a sentence structure with precise grammatical explanation. The „strange" ones are there to combine with the others in order to correct the structures of the previously applied rules. Although some of the rules are obvious and we believe that a human would derive the same rules, still the grammatical meaning of the rules can be evaluated only when analyzing the result of the application of the whole list of rules. Let us try and examine the relation and dependence of the reduced and non-reduced tagset on the selected rules as given in the above tables. One of the obvious rules in Table 3.4 is rule 1 (and rule 6; the rules might repeat; during their repetition their influence has a different scope depending on their position in the list). To cope with a more distinct situation in a more specific POS TAG1171, the algorithm produces more rules in order to capture the relation between an adjective and a noun: rules 2, 6, 7, 9, 10 and 11 in Table 3.3.

Undoubtedly, the reduced tagset brings better absolute values. We would like to note that the nodes within the syntactic structure could be returned to the corresponding values from the full tagset. Thus, no information has been lost.

As for the rule-based application for syntactic structure extraction, the reduced tagset leads to a much better start on the learning curve (Figure 3.2). After the saturation of the process of learning, which comes after several tens of rules have been learnt, the learning might continue after one switches to the more expanded tagset, namely the full tagset.

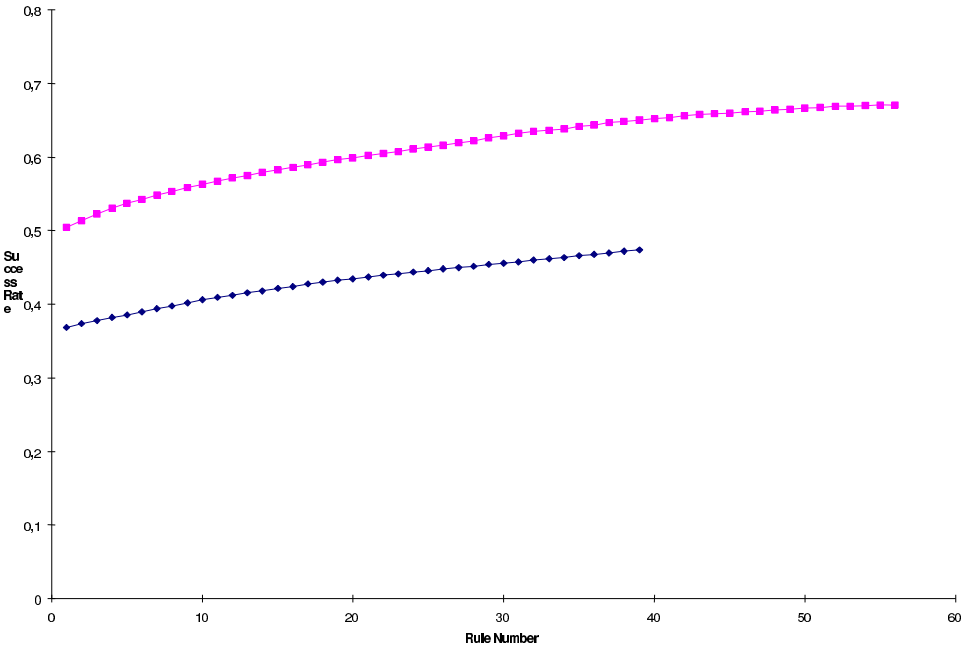


Figure 3.2: RBA learning curves

Being sensitive to the tagset size, the rule-based approach operates better on smaller tagsets. A reduction of a tagset should not distort the syntactic characteristics of the input. The type of the reduction is dependent on the tagset characteristics.

3.3. Linguistic recommendations for morphemic tagset modification for syntactic parsing

The following are some linguistic recommendations for morphemic tagset modification (mainly reduction) of Czech for syntactic parsing.

For all POS, the 1st and the 2nd positions (the POS and the sub POS) are preserved.

For nouns, adjectives, pronouns, numerals: the gender, number and case categories (3rd, 4th and 5th position) are preserved.

In case of adjectives in nominative, two different syntactic relations are possible: attributive (as in "That beautiful girl is my daughter.") and predicative (as in "That girl is beautiful."). These two cases are to be distinguished by the parser; the Czech morphology does not distinguish them.

Numerals and pronouns may be selectively merged with nouns and adjectives (in the case of numerals also with adverbs).

For pronouns one may, for the purpose of syntactic analysis, make the following merging (the "(!)" sign marks that the distinction is not a part of the morphemic tag):

a) the following pronouns can be treated as nouns (NN): personal pronouns, pronoun dem., pronoun rel. (as něhož, němuž, ...) marked as PJ, pronoun rel. genderless, pronoun quest. or rel. (!), indef. pronoun (!), negative pronoun genderless (!).
b) the following pronouns can be treated as adjectives (AA): possessive pronouns, rel. pronoun marked as P1, reflexive possessive pronoun, pronoun self, quest. or rel. pronoun (!), indef. pronoun (!), neg. pronoun (!)
c) reflexive pronouns should not be merged, and will remain as pronouns.

For numerals one may, for the purpose of syntactic analysis, make the following merging:

a) adjective numerals can be treated as adjectives (AA).
b) multiplicative numerals can be treated as adverbs (DB)
c) all other numerals can be treated as nouns (NN)

For the case of verbs and the purpose of syntactic analysis, due to the relevance of the verb frame information (which is not explicitly a part of this approach), we suggest to preserve verb's POS and sub POS and the following categories of person, number, mode, tense and mood.

If the tagset permits, it is advisable to pay a special attention to the verbs "mít" (to have) and "být" (to be). As for the verb "být" in the sentences where more than one verb occurs, more detailed distinctions should be made, for:

- "být" as an auxiliary verb in future tense
- "být" as an auxiliary verb in past tense
- "být" as an auxiliary verb in a conditional clause
- if no other verb form occurs, then "být" as a copule or verb of existence.

Therefore, the 'tag' for these two verbs will be constructed as their lemma and 'full morphological tag'. If the system architecture is able to handle a larger tagset, it is advisable to extend (instead of reduce) the tagset with verb lexemes, e.g. the first 100 most frequent verbs.

Within sentences with more than one verb a distinction between coordinating and subordinating conjunctions should be made. We preserve all distinctions for conjunctions, as originally in the tagset.

As verbs, also prepositions deserve special attention. We preserve their case category. We would also like to extend the tagset with preposition lemmas in the similar way as we stated above for verbs. A prior analysis of relations between preposition lemmas and possible case bounds should be made (e.g., the preposition "na" may be followed only by Accusative or Locative case; the preposition "za" may be followed by the Genitive, Accusative or Instrumental case; etc.) and this knowledge should take part of the tree transformation process.

All adverbs will be represented by a single tag DB. Interjections and particles tags are preserved is in the original tagset, i.e., without any further distinctions using tags T and I respectively.

3.4. On rule modifications

There are no directions according to which a rule is designed. Nevertheless, there are several guidelines the rules follow:

- The form of the rule is limited from the needs of the operations to be performed, thus from the aim to be achieved.
- The rule should be computationally achievable within the time and space determined by the application needs.
- The rule is suited to the formal framework of the language theory. It is the design of the templates of the rules which are language dependent although the RBA as such is language independent.

The first set of rules applied for Czech (Chapter 1) followed the logic of the rules for English (Brill 1993a,b). The latter, performed (conditioned under various triggering environments) only two simple actions: addition or deletion of constituency boundaries resulting in their left or right shift, as summarized in Figure 3.3.

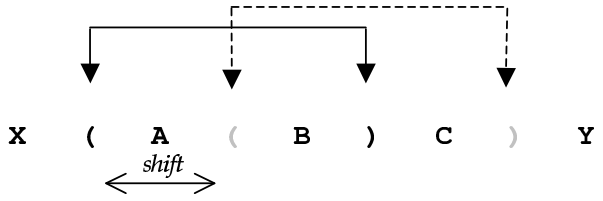


Figure 3.3: RBA for constituents

As showed in the previous sections, such rules do not guarantee successful training. A new exhaustive set of rules was designed. Those were designed such that they cover the basic operations that can be performed on a dependency tree. The interpretation of the rules was modified so that they operate on the tree

representation of the sentence: searching for a left paren between A and B, such is located also in the linear representation A (C, B), since the left paren is understood as a dependency. Thus, the changes the rule performs are always within a distance of one tree level[15].

In the following, a list of new rule templates is presented, the rules are explained and their performance is exemplified. For the rules which add a dependency it is necessary to check first that the dependency one wants to add does not already exist; for the rules which delete a dependency it is necessary to check first whether the dependency to delete exists; analogously, similar checks are done on the triggering environment for each rule before its application.

Rule	Explanation
ADD (LEFT A	Make A to be the governing node of the brother nodes of A
ADD (RIGHT A	Push A one level lower in the dependency path
ADD (BETWEEN A B	Add a dependency between A and B if both of them are brother nodes.
ADD) LEFT A	Locate a first possible corresponding "(" and apply to it the "ADD (" rule.
ADD) RIGHT A	Locate a first possible corresponding "(" and apply to it the "ADD (" rule.
ADD) BETWEEN A B	Locate a first possible corresponding "(" and apply to it the "ADD (" rule.
DEL (LEFT A	Move the subtree represented by A up in the tree structure by attaching it to the governor of A.
DEL (RIGHT A	Move all subtrees of A on the same level (attached to the governor of A) of A.
DEL (BETWEEN A B	If A is a governor of B, perform "DEL (RIGHT A".
DEL) LEFT A	Locate a first possible corresponding "(" and apply to it the "DEL (" rule.
DEL) RIGHT A	Locate a first possible corresponding "(" and apply to it the "DEL (" rule.
DEL) BETWEEN A B	Locate a first possible corresponding "(" and apply to it the "DEL (" rule.
SWAP (RIGHT A	For all such X, where A is a governor of X, perform "SWAP (BETWEEN A X".
SWAP (LEFT A	For all such X, where X is a governor of A, perform "SWAP (BETWEEN X A".
SWAP) RIGHT A	Move A to the subtree to the right at the same tree level (if any) as A is.
SWAP) LEFT A	Move A to the subtree to the left at the same tree level (if any) as A is.
SWAP (BETWEEN A B	If there is a dependency relation between A and B change its direction from AB to BA.

[15] The complexity of the process of learning would significantly increase if rules operating on nodes not being directly dependent are used.

SWAP) BETWEEN A B	If there no dependency relation between A and B and B is one level higher than A, swap A and B.
SWAP COMMA BETWEEN A B	If A and B are on the same tree level exchange between them the subtrees they govern.
MAKE ROOT LEFT A	Make node A to be the root of the tree.
MAKE ROOTFIRST BETWEEN A B	If A and B are brother nodes, A becomes a root node.
MAKE ROOTSECOND BETWEEN A B	If A and B are brother nodes, B becomes a root node.
MAKE GROUP LEFT A	All nodes governed by A becomes immediate brother nodes of that node (the structure collapses such that everything is governed by A).
MAKE GROUPFIRST BETWEEN A B	If A and B are brother nodes, all nodes governed by A becomes immediate brother nodes of A.
MAKE GROUPSECOND BETWEEN A B	If A and B are brother nodes, all nodes governed by B becomes immediate brothers of B.
MAKE DEPENDENCE BETWEEN A B	If there is no dependency relation between A and B, establish it. A and B can be anywhere in the tree (exception to the 1 level distance application of the rules).

Table 3.5: Second set of modified rules

The last seven rules were later excluded since they were never selected as best rules.

Some general remarks on the rules from Table 3.5:

- all of the rules follow the form as presented earlier (Chapter 1, Section 1.2): `<action> <what> <where>`
- LEFT means 'to the first suitable position to the left' in the linear form of the tree
- RIGHT means 'to the first suitable position to the right' in the linear form of the tree
- Each sentence is represented in its linear form. The rules operate on the linear form but their interpretation is as if they would operate on a tree.
- As for the transformation A and B represent the whole subtrees they govern.
- There are no limitations whether the rules can or cannot create non-projective syntactic trees, nevertheless the rules are not designed to identify them and treat them in a separate manner.

In order to illustrate the application of the main types of the rules, let us take the following linearized tree structure from Figure 3.4.

```
A ( B( C, D), E( F, G), H, I, J( K (L, M))).
```

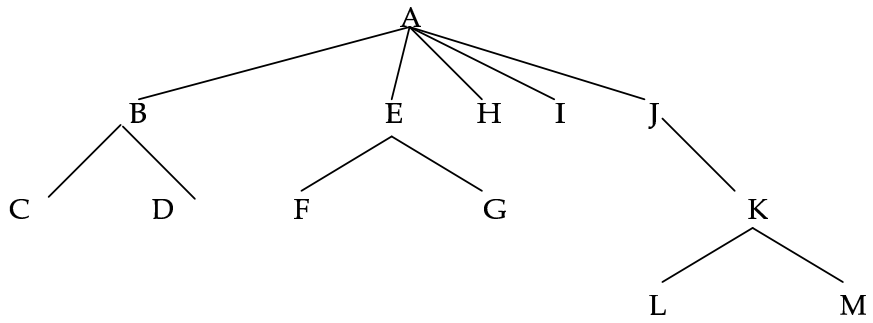


Figure 3.4: Abstract dependency tree structure

If the rule `ADD ) LEFT D` is applied to the tree from Figure 3.4 the resulting structure will be

```
A ( B( C), D, E( F, G), H, I, J( K (L, M) ) )
```

as in Figure 3.5.

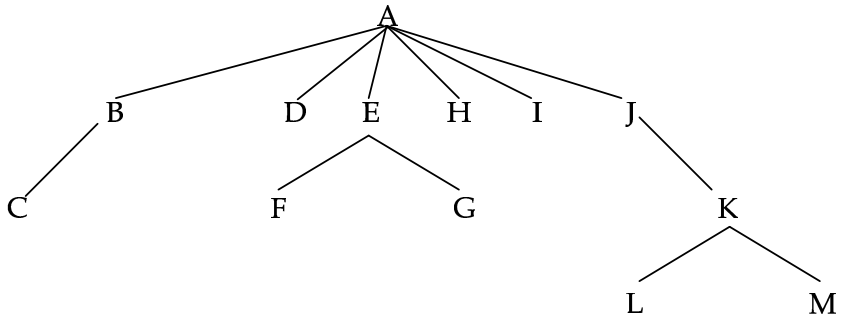


Figure 3.5: Application of ADD) LEFT D

`SWAP COMMA BETWEEN G K` changing the original tree from Figure 3.4 into

```
A ( B( C, D), E( F, K (L, M)), H, I, J( G ) )
```

as in Figure 3.6.

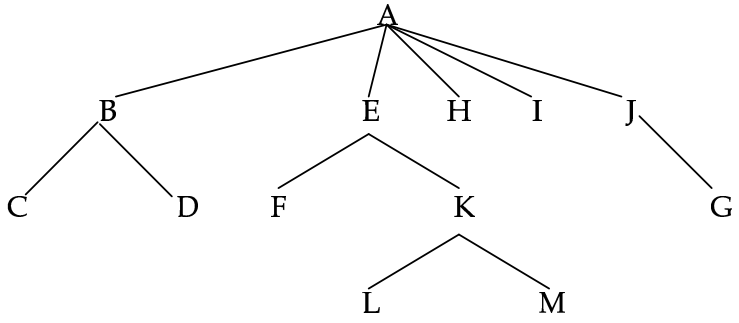


Figure 3.6: Application of SWAP COMMA BETWEEN G K

The rule `SWAP ( BETWEEN J K` makes the following changes to the tree from Figure 3.4

```
A ( B( C, D), E( F, G), H, I, K( J, L, M ) )
```

as presented in Figure 3.7.

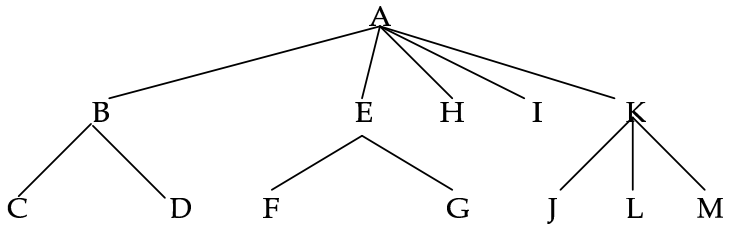


Figure 3.7: Application of SWAP (BETWEEN J K

Finally, `ADD ) LEFT I` adds the right bracket which means moving all of the brother nodes to the left from node `I` one level deeper within the dependency structure in Figure 3.4, which results into

```
A ( B( C, D), E( F, G, H) I, J( K (L, M) ) )
```

as presented in Figure 3.8.

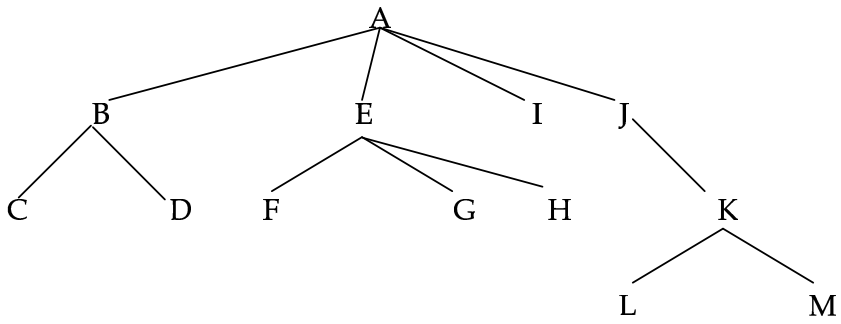


Figure 3.8: Application of ADD) LEFT I

The rest of the rules can be directly deduced from the above mentioned transformations.

As mentioned earlier, the last seven rules from Table 3.5 did not improve the success rate while parsing and therefore, they are not exemplified here. Nevertheless, one could easily imagine the tree transformations they perform.

3.5. The error function

The error function is the key determiner of how good certain structure is when compared to another one. The error function used in all of the experiments with syntactic dependency structures of Czech (not only for RBA) in all of the works presented so far is a ratio between the correct and the total number of dependencies. If not enough attention is paid, such an error function will reflect no difference between tree structures from Figure 3.9 in terms of the values in their nodes (if only the node values as morphemic tags or lexical items are included) .

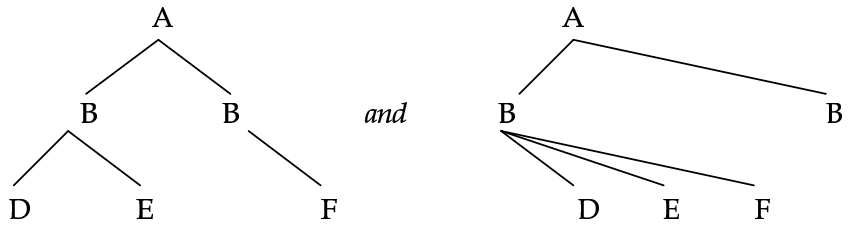


Figure 3.9: Different evaluations

There are other situations where such error function would be the bad guide for the process of learning. If a dependency relation A - B (Figure 3.10(a)) has to be compared to the dependencies A - C and C - B of a tree path A - C - B on one hand (Figure 3.10(b)), and on the other hand to the dependencies C - A and C - B of the subtree C (A, B) (Figure 3.10(c)), obviously the counting of the success rate (as introduced before) will yield 0% in both of the cases. But, it seems that there is a dependency (non-direct) relation between A and B in the first case, and that one should be able to make a difference between the first and the second case.

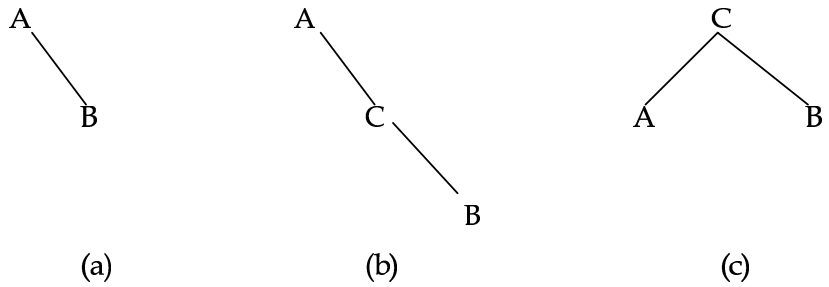


Figure 3.10: Path evaluation

For an automatic transformation to be performed by the stated rules it is "easier to make" B directly dependent on A at Figure 3.10(b) since B is in the subtree of A, while it is more difficult to achieve the required dependency in the case of Figure 3.10(c) since B and A are two different subtrees which are not necessarily next to each other. Also at Figure 3.10(b) it can be considered that B is not directly dependent but remains to be at least distantly dependent on A.

If the success of the algorithms used for dependencies is to be compared to the one used for constituency, using the mentioned error function would make it neither directly nor approximately possible. Assuming that a constituency can be determined by its head the non terminal nodes can be lexicalized. Each terminal node is determined by the path it follows from the root of the tree. Far from linguistically similar, paths can be listed also from the dependency tree. Thus, comparing paths instead of dependencies can make the absolute success rates more comparable.

Therefore, this 'path evaluation' approach would consist of including the length of the dependency (the direct dependency could have the value of 1, while the non-direct dependency could be given a success rate as a decreasing function of the path length). Two functions are directly offering themselves to fulfill this

aim: reciprocal distance $f(n)=\frac{1}{n}$ and negative exponential distance $g(n)=\frac{1}{e^{n-1}}$. The latter "punishes" the non-direct dependency distance more than $f(n)$ does.

For the previous comparison example, A - B to A- C - B, the direct dependency A - B exists as a non-direct dependency in A - C - B a "weight" or "success" of $g(n=2)=\frac{1}{e}\cong 0.37$, opposed to 1 if A-B had to be compared to A-B.

In order to be able to calculate the success rate a $V(T, F)$, the *value* of a tree T should be defined, which would allow for the calculation of the ratio of existing over possible values (precision). Let $V(T,F)=\sum_{\forall path\ of\ T} F(path)$, where $F(n)$ is either $f(n)$ or $g(n)$.

If T is a tree describing a correct syntactic structure then its value is $V(T, F)$, which will always be greater than any incorrect variant.

Let us compare the structure T_1: A (B(C(D))) to the structure T_2: A(B(D),C). Let the latter be the true structure. Its value is $V(T_2, g) = 3 + 0.37 = 3.37$. If T_1 is to be compared to T_2 one may compare any of the possible paths in T_1 (A - B, B - C, C - D, A - B - C, B - C - D, A - B - C - D) to all of the paths of T_2 (A - B, B - D, A - C, A - B - D). The set of comparable paths is $T_{1,2}$(A - B, A - B - C, B - C - D) with its value $V(T_{1,2}, g) = 1 + 0.37 + 0.37 = 1.74$.

The success rate is then $V(T_{1,2}, g)/ V(T_2, g) = 1.74/3.37 = 51.63$ %.

The use of the path evaluation function instead of the classical one helped the learning process of the RBA approach, where both $f(n)$ and $g(n)$ supported the learning process relatively better than the classical evaluation function.

All over this work, this is the only place where different than dependency accuracy measure is implemented (also used for finding the best rules presented in Table 3.7).

3.6. Learning discipline

A rule is applied to a sentence as many times as possible. If once selected, there is no limitation according to which the rule could not be selected next time.

The search remains the greedy one.

3.7. Success and conclusion on the RBA modifications

The changes stated above result in following best average success rates:

- of 65 % while learning
- of 43 % while parsing (using the 'classical error function')

The list of the best rules learned from 100 sentences is presented in Table 3.6. The tagset has been reduced and only the first two characters (positions, POS and sub POS) are taken into consideration. The success rate presented in Table 3.6 is the cumulative success rate of the rule during the process of learning. Experiments

with more than 100 sentences (200 and 300 sentences) were also carried out. None of those experiments resulted in a list of more successful best rules than the one presented here.

Rule	Success		
SWAP (RIGHT AF	0.415543	ADD (BETWEEN VP NM	0.594766
SWAP (LEFT JE	0.436162	DEL) LEFT NM	0.595559
SWAP (BETWEEN ZI JS	0.448850	SWAP (BETWEEN JE RV	0.596352
SWAP (RIGHT PS	0.458366	SWAP (BETWEEN JS PD	0.597145
SWAP (RIGHT PQ	0.467883	ADD (BETWEEN ZI NO	0.597938
SWAP (RIGHT DB	0.477399	DEL (RIGHT PA	0.598731
SWAP (RIGHT PD	0.486122	ADD (BETWEEN NO ZI	0.599524
SWAP (RIGHT DG	0.493259	ADD (BETWEEN VM NF	0.600317
SWAP (BETWEEN ZI VP	0.500396	ADD (BETWEEN ZN AB	0.601110
DEL (RIGHT AF	0.505155	SWAP (BETWEEN DB AV	0.601903
SWAP (RIGHT PR	0.509913	SWAP (BETWEEN VP VS	0.602696
SWAP (RIGHT VC	0.515464	SWAP (BETWEEN AF NO	0.603489
ADD (BETWEEN R4 NF	0.519429	ADD (BETWEEN NO AF	0.605075
SWAP (BETWEEN ZI JE	0.523394	SWAP) BETWEEN AF NF	0.606661
SWAP (BETWEEN ZN JS	0.527359	SWAP COMMA BETWEEN ZI PI	0.607454
SWAP (BETWEEN PP VP	0.530531	DEL (RIGHT PL	0.608247
SWAP (RIGHT NO	0.533703	SWAP (BETWEEN NF VR	0.610626
ADD (RIGHT R2	0.536875	SWAP COMMA BETWEEN NI NI	0.611419
SWAP (RIGHT AM	0.540048	ADD (BETWEEN NI DB	0.612213
DEL (RIGHT RV	0.542427	ADD (BETWEEN NI R4	0.613006
DEL (BETWEEN ZI NF	0.544806	ADD (BETWEEN NM AB	0.613799
SWAP (BETWEEN NI DB	0.547978	SWAP (BETWEEN NI RV	0.614592
DEL (RIGHT T|	0.550357	SWAP (BETWEEN R2 RV	0.616178
DEL (BETWEEN DB DB	0.552736	DEL (BETWEEN RV R3	0.616971
SWAP (BETWEEN JE VV	0.555115	ADD (BETWEEN R4 NN	0.617764
SWAP (RIGHT AI	0.557494	SWAP (BETWEEN R4 R4	0.618557
SWAP (BETWEEN VF VR	0.559873	ADD (BETWEEN DG PD	0.619350
ADD (BETWEEN PD DB	0.561459	SWAP (BETWEEN CG NF	0.620143
SWAP (BETWEEN JS NM	0.563045	DEL (BETWEEN JE PD	0.620936
DEL) RIGHT ZI	0.564631	ADD (BETWEEN JE AF	0.622522
ADD (BETWEEN RV NI	0.566217	ADD) BETWEEN PQ NF	0.623315
DEL (BETWEEN JE R7	0.567803	SWAP (RIGHT VU	0.624108
SWAP COMMA BETWEEN VP NF	0.569389	DEL (RIGHT PN	0.624901
SWAP (RIGHT PE	0.570975	ADD) BETWEEN VP NF	0.625694
SWAP (BETWEEN NF VR	0.572562	SWAP (BETWEEN PQ JS	0.626487
SWAP (BETWEEN VM NO	0.574147	ADD) BETWEEN AV NM	0.627280
SWAP (BETWEEN VM VR	0.575734	DEL (RIGHT AC	0.628073
ADD (BETWEEN JS VP	0.577320	ADD (BETWEEN R3 NF	0.628866
DEL) BETWEEN VR ZI	0.578906	ADD (BETWEEN VF R4	0.629659
DEL (RIGHT DG	0.580492	DEL (BETWEEN VF AF	0.630452
SWAP (BETWEEN JE JS	0.582078	ADD (BETWEEN VF JE	0.631245
ADD (BETWEEN VP NI	0.583664	DEL (BETWEEN NI R3	0.632038
DEL (BETWEEN ZI NM	0.585250	DEL (BETWEEN NM NM	0.632831
SWAP (BETWEEN JE ZS	0.586836	SWAP (BETWEEN R2 ZI	0.636003
SWAP (BETWEEN T VP	0.588422	ADD (BETWEEN VP NM	0.636796
ADD (BETWEEN JE AF	0.589215	SWAP (BETWEEN ZI CX	0.637589
ADD) BETWEEN DB R4	0.590008	SWAP (BETWEEN JE VS	0.638382
SWAP (BETWEEN ZI VG	0.590801	SWAP (BETWEEN DB JE	0.639968
ADD (BETWEEN JE NF	0.591594	SWAP (BETWEEN VF VS	0.641554
ADD) BETWEEN AF DB	0.592387	ADD (BETWEEN R2 NN	0.642347
SWAP (BETWEEN PD DG	0.593180	DEL (BETWEEN ZI RV	0.643140
SWAP (BETWEEN DG VP	0.593973	SWAP (BETWEEN VP VR	0.643933
		DEL (BETWEEN NI AB	0.644726

Table 3.6: List of best rules learned on 100 sentences

If the rule-based error driven learning includes this error function, the rules presented in Table 3.7 can be achieved (as in the previous case the learning process was on 100 sentences; the same set of sentences as for the best rules of Table 3.6).

Rule	Success
SWAP (RIGHT AF	0.590218
SWAP (RIGHT ZI	0.613243
SWAP (LEFT JE	0.630062
SWAP (BETWEEN PQ VP	0.648547
SWAP (LEFT VR	0.658814
SWAP (RIGHT DB	0.668000
SWAP (RIGHT DG	0.676885
SWAP (RIGHT PS	0.684720
SWAP (RIGHT PD	0.691009
SWAP (RIGHT PR	0.697009
SWAP (BETWEEN ZI DB	0.701609
DEL (RIGHT PP	0.705621
DEL (BETWEEN NI ZI	0.709475
SWAP (BETWEEN NI JE	0.713570
DEL (BETWEEN RV JS	0.716804
DEL (BETWEEN RV R2	0.719927
SWAP (BETWEEN RV VP	0.722952
DEL (RIGHT PQ	0.725858
SWAP (BETWEEN NI ZI	0.728761

Table 3.7: A fragment of the list of best rules obtained from the learning process on 100 sentences with negative exponential error function

The success rate of the learning process is higher since the way it has been calculated results in higher percentage rates. This higher success rate does not necessarily mean a less erroneous output.

'Playing' with the rules has been a long and tedious process, which results in the following observations:

1. Once the rule templates are postulated, it is advisable to test them on various sentences, one sentence (or a small set) at a time. Good set of rules should be able (up to justifiable exceptions) to learn the complete parsing structure of the single sentence. This could be a-which-rules-to-take 'criterion'.
2. Parsing should be done with the smallest possible tag set, but such that will reflect the syntactic relations of the language.
3. Saturation of the learning process happens rather fast. Therefore, it is important that the initiating tree structure has a higher success rate. E.g. the right chain structure would not be the most suitable one.
4. The best rule to select in the next iteration would be most probably a rule which performed similarly well as the best rule had performed in the current iteration. This is a useful heuristics that could significantly speed up the learning process. On other rule characteristics and tracing of rule dependence see the contribution of L.A. Ramshaw and M.P. Marcus in (Klavans, Resnik 1996).
5. The answer why did not the modified RBA perform better can be formulated as a hypothesis that a similar learning processes should distinguish two phases:
   a. Diminishing the influence of the surface word order, and afterwards

b. Learning the syntactic structure.

Ad 5., while the learning of the syntactic structure could be done within the same approach as presented in this chapter, diminishing the influence of the surface word order is a motivating element for the studies in the following chapters, where the so-called sentence graphs are employed.

3.8. Tagging with analytical functions

In the previous sections a parsing structure without the syntactic analytical functions (afuns) was assigned to the input morphologically parsed sentence. If a syntactic tree without afuns is achieved then experience from the manual and semi-automatic annotation of the treebank structures and functions shows that it is possible to assign the afuns to the tree nodes automatically with a high success rate.

On the other hand, as described in Section 3.2, the tag set size influences the success rate of a tagging procedure: the smaller the tagset size the higher is the chance to learn the regularities of the language with a higher success rate. As we will see here, the tag set size is not the only 'parameter' that influences the success rate. In fact, it is the 'logic of application' of the tags that is of primer importance.

Assuming that each lexical item has its own analytical functions (afun), as it has a morphemic tag, it is possible to try to tag an input text using the language independent tagging procedures (used for morphemic tagging) for tagging with afuns. Hopefully, having the input structure tagged correctly by afuns, obtaining the syntactic tree structure would be an easier problem since afuns already contain syntactic information.

In this section I will describe the experiment of tagging an input text by afuns. For matter of convenience, the tagger used is a Rule-based tagger (Brill 1993c).

The following are the input facts:

- Tagging with tags from a tagset of 1171 morphemic tags yields a success rate of approx. 80%, while tagging with tags from a tagset of 206 morphemic tags yields a success rate of approx. 87%. (as in Section 3.2). The state of the art for morphemic tagging of Czech on its full tagset is currently approx. 95%.
- The tagset of afuns consists of 70 tags.
- Rule-based tagger claims no dependence on a tagset.

If applied as such and tested, without using any additional information, the rule-based tagger (trained on the pair: lexical token, afun) yields a success rate of approx. 66% on the test set for the analytical functions. The rule-based tagger was selected for matter of convenience and relatively fast and easy training process. The aim is to get relative observations not absolute best numbers. The presented results were not further improved.

Since the syntactic level is superordinate to the morphemic one, and in order to trace the reason for such a low success rate (when compared to the

success rate when the rule-based tagger is used in order to tag the tokens by their morphemic tags), in the next experiment, information of the morphemic level, i.e. the morphemic tags, was included. At this step, the lexical token has been substituted by its morphemic tag. Thus, the rule-based tagger was trained on the pair (disambiguated morphemic tag, analytical function).

In this case a success rate of 72% was recorded, which is 6% improvement compared to the 66% when the morphemic information was not taken into consideration.

Both of the experiments were provided on the same training sets, and tested on the same test sets, hence the results are directly comparable.

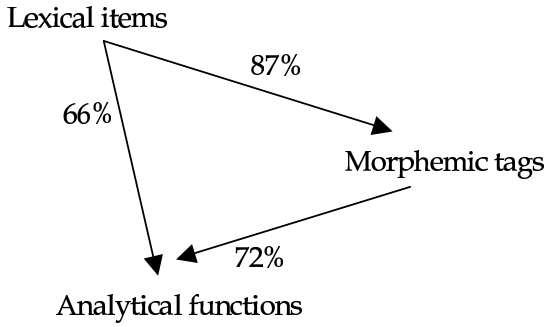


Figure 3.11: The afun triangle

The success of tagging by afuns is very low in order to use it as input for a tree structure determination. Although low, the results of tagging by afuns are significant for the parsing-by-tagging experiment setting in Chapter 8.

3.8.1. Conclusion

Without performing the experiments it would be difficult to expect such results, since it would be expected that the usage of the morphemic information would contribute more significantly to the success rate of tagging of the analytical functions.

One may conclude that:

- The success rate of the rule-based tagger depends not only on the cardinality of the tag set but also on its structure. Hence, the claim that a higher success rate is due to low cardinality of the tagset, when success rates of the same task but over different languages are being compared, is not sufficient.
- The rule-based tagger, as originally proposed in (Brill 1993c), cannot be successfully directly applied for a successful tagging of analytical functions.
- One of the basic reasons, as shown by analysis of PDT, is in the different nature of ambiguity present within analytical functions. Other reasons include, as Czech is a free word order language, that it is frequently not the case that the immediate neighboring tokens determine the value of the token to be tagged.

- Since the rules operate on close neighboring environments, those environments are not big enough to capture the afuns. This supports the hypothesis stated in Section 3.7 that the word order, i.e. sentence structure, plays an important role in NLP of Czech and is one of the main reasons that influences the success of the automatic procedures. In fact, all of the procedures were originally designed for English and modified for Czech.

3.9. Summary on the RBA modifications

I would like to summarize here all modifications of RBA stated in the precious chapters in order to support a claim that almost *all possible modifications of RBA* have been exhausted.

The RBA performance depends on the following elements: the tree initial structure, the way of obtaining the best rules, the error function, the rule templates, the cardinality of the tagset, the size of the train set, the sentence length. One by one, I will stress the main considerations with respect to the RBA performance.

1. *The tree initial structure*: (Brill 1993c) demonstrates that RBA is sensitive to initial structures and experiments with random initial structures or binary tree initial structures perform worse than the right chain one. The initial structure can also be an output of another algorithm. This approach was tested (although not documented here), but did not yield any significant improvement. In the latter case, such non-trivial input structure would require different and more sensitive set of rules. Therefore, the right chain initial tree structure was selected to be the best one for study of the RBA adequacy of parsing of Czech.
2. *The way of obtaining the best rule*: As stated in (Klavans, Resnik 1996) and supported by a variety of studies on the performance of RBA, employment of other than the greedy search method does not yield any significant improvement. The article in the mentioned book edited by Klavans and Resnik, has tested backtracking search to certain depth, and N-best search. It was not demonstrated that those could perform better than the greedy search. While observing the best rule selection, it could have been noticed that the next best rule to greedy-select (in iteration t+1) in the majority of the cases (specially before the saturation of the learning process) would be the one from the set of closest (in terms of success) rules to the best rule in iteration t. This observation is also in accordance with the claims from (Klavans, Resnik 1996).
3. *The error function*: A different, more indicative, error function type was used for the process of learning (Section 3.5). The RBA performed slightly better. It was shown that the error function while learning should not necessarily be the error function used for evaluation.
4. *The rule templates*: In Chapter 1 and in this one there is a variety of rule templates tested. Rules dealing with brackets and commas dependent of the

linear tree description were tested in Chapter 1, while rules operating on the linear tree structure but with tree structure semantics on +/- 1level of the syntactic tree were demonstrated in Section 3.4. The set of rules, as in all published experiments of RBA, cover a complete set of elementary operations of the tree structure. The performed modifications apply a complete set of elementary tree changes.

5. *The cardinality of the tagset*: All basic experiments with tagset modifications were performed, which main part is summarized in Section 3.2. It was shown that the robustness of RBA is big and that RBA performs better on smaller tagsets. Recommendations on linguistically relevant tagset reductions is presented in Section 3.3.
6. *The size of the train set*: Experiments were performed with train sets of 100, 200 and 400 sentences. It was shown that the best set of rules was obtained not from the larger sets but from the 100 sentences train set. Based on such results there was no need of testing on larger train sets.
7. *The sentence length*: The shorter the sentence is, the more successful the parsing is - is a common claim for the RBA approaches and valid also for our case. Experiments were done with train sets of short sentences (less than the average length) and as well, experiments were performed on sentences with one, two or different number of verbs. There was improvement of learning but not such to be presented as a direction to follow for these cases.

With respect to the above truths and we respect to the success rates as presented in Section 3.7 and summarized in Subsection 3.9.1 it can be concluded that the rule templates and the RBA strategy of changing the dependency tree structure is not a direction to follow. The statistics, e.g. for dependency length distribution (Chapter 2, Table 2.3) indicate that the surface ordering is not of a local character, which is contrary to the local character of changes done on the tree structure: a challenge to overcome.

Hence, the answer should be searched somewhere else, and studies performed in the following chapters demonstrate that. Chapter 5 is motivated by the aim to get closer to the effect of syntactic vicinity of words.

3.9.1. The percentages of RBA

Table 3.8 presents a summary of the success rates presented in this chapter and it also includes the first RBA results presented in Chapter 1. If not otherwise stated, the sentences in the test or train data sets have been selected randomly.

Algorithm	Size and identification of train data	Size and identification of test data	Dependency accuracy	Note
RBA, Chapter 1, *With no treebank available all sentences were hand crafted only for the purpose of development of the algorithm.*	25 sentences	20 sentences	42%	
	100 sentences	*the same as above*	at most 42%	
RBA, Chapter 3, *PDT was under development; the used sentences at that time consisted of many linguistic inconsistencies (cleaned several years afterwards)*[16]*. During that period the morphemic tagset has also been changed several times before it settled to the positional one.*	100 sentences	on the train data	47%	On $PosTag_{1171}$ as in Table 3.3.
	the same as above	on the train data	65%	On $PosTag_{34}$ as in Table 3.4. The same accuracy was observed on modified positional tagset, taking only the first two positions, i.e. POS and subPOS..
	100 sentences *[1]*	400 sentences	43%	On POS and subPOS only.
	200 sentences	*the same as above*	slightly less than 43%	On POS and subPOS only.
	300 sentence	*the same as above*	slightly less than 43%	On POS and subPOS only.
	100 sentence, *the same as [1] above*	on the train data	73%	On POS and subPOS only, using error function as in Section 3.5.
	100 sentences with 1 verb	200 sentences with one verb	47%	On POS and subPOS only.
	100 sentences	200 sentences	56%	No morphemic but analytical function tags were used[17].

Table 3.8: The percentages of RBA

[16] Notes on the development of the algorithm with respect to time in years of development are in Chapter 9.

[17] This experiment was performed only for the purpose of verification of the abilities of the rules to capture syntactic information. Due to the low level of accuracy of tagging by analytical functions as in Section 3.8, there was no reason to continue with these type of experiments.

4. The Directed Minimum Spanning Tree

This chapter is devoted to a description of an algorithm for rooted directed minimum (as well maximum) spanning tree. To my best knowledge, this is the first time such algorithm is applied for parsing, as demonstrated in the following chapters.

Consider a weighted directed graph G(V,E), where V is the set of vertices, E is the set of edges, w(*i,j*) is a weight of the edge *(i, j)*, $i, j \in V$ directed from *i* to *j*, $n=|V|$ and $m = |E|$. Let $T(V,E_T)$ be a rooted directed tree, where E_T is a subset of E. Let S_{E_T} represent the sum of weights of all edges of E_T. $T(V,E_T)$ is a *rooted directed minimum spanning tree* (MST) of the graph G(MST(G, R), V ∋ R is the root) if there no other $T'(V,E_T)$, where $S_{E_{T'}} < S_{E_T}$. The rooted directed spanning tree is a graph, which connects without any cycle all *n* vertices with *n-1* edges, i.e., each vertex except for the root has one and only one incoming edge.

Obtaining MST(G,R) is not a straightforward problem. On the other hand obtaining a MST of an *undirected* graph is well known one with several algorithms for it: the Prim's (in the Czech literature sometimes called Jarník's algorithm), the Kuskal's, and the Borůvka's one (Matoušek, Nešetřil 2000). While the Prim's and very often also the Kruskal's algorithm can be found in almost any even introductory text of the theory of graphs, the 'similar' problem for directed graphs is included neither in more advanced texts of graph theory. And, it is not the case that the undirected versions can be applied for directed graphs, which can be demonstrated for the Prim's algorithm on the graph from Figure 4.1.

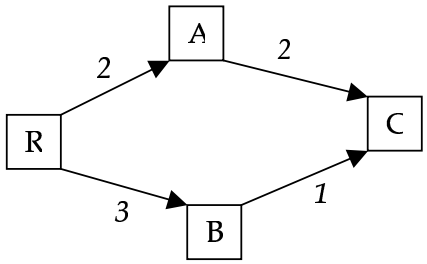


Figure 4.1: Prim's algorithm for oriented MST

Trying to find the directed rooted MST of the graph from Figure 4.1 using the greedy Prim's algorithm would result in selecting (R, A), (A, C) and (R, B) with a total weight of 7, while it is obvious that there is another directed rooted MST with a lower weight: (R, A), (R, B) and (B, C) has a total weight of 6.

A solution to the directed rooted MST was given independently by Chu and Liu (Chu, Liu 1965), Edmonds (Edmonds 1967) and Bock (Bock 1971). All of them efficient and mutually similar, Block presents the solution on matrices, while the other two present the solution on graphs. In the sequel, I will present

the Chu-Liu/Edmonds algorithm[18] for directed rooted MST and explain it on an example as in (Lawler 1976).

Chi-Liu/Edmonds Algorithm

1. Let r is the root vertex. Discard all arcs entering r.
2. Form the following set
   $S = \{ (i, j) \mid (i, j)$ is the entering edge with the smallest cost for all $i \neq r\}$
3. If no cycle is formed G(V, S) is MST(G, R), otherwise continue.
4. For each cycle formed, contract the cycle into a pseudo-vertex k and modify the weight of each edge which enters vertex j in the cycle from some vertex i outside the cycle according to the following formula
   $$w(i,k) = w(i,j) - \left[w(x(j), j) - \min_j \{w(x(j), j)\} \right],$$
   where $w(x(j), j)$ is the weight of the edge in the cycle which enters j.
5. For each pseudo-vertex, select the entering edge, which has the smallest modified weight from step 4. Replace the edge, which enters the same real vertex in S by the new selected edge.
6. Go to step 3.

In order to eliminate cycles, the algorithm tries to find a 'replacing' edge, which has the minimum extra weight. The formula from step 4 defines this extra weight.

Figure 4.2 and Figure 4.3 illustrate the contracting (pseudo-vertex) technique. The root node is node 1.

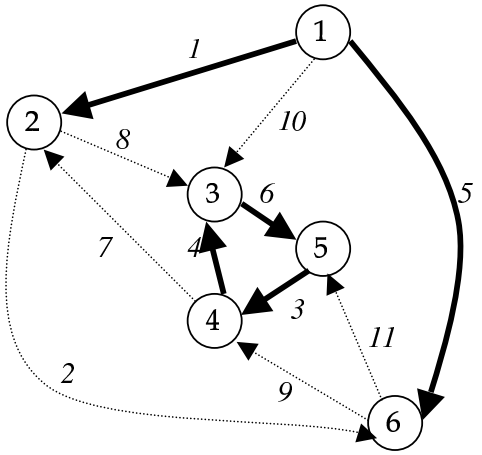


Figure 4.2: MST graph example

[18] For an efficient implementation ($O(m\log n)$ and $O(n^2)$ for dense graphs) see (Tarjan 1977) and (Camerini, Fratta, Maffioli 1979).

The set S in Figure 4.2 is $S = \{(1,2), (1,6), (4,3), (3,5), (5,4)\}$

After an application of steps 3 and 5, the set S becomes $S = \{(1,2), (1,6), (2,3), (3,5), (5,4)\}$. It can be noted that there is a new edge $(2,3)$ with its minimum extra weight. This edge replaces the edge $(4,3)$ in S.

With respect to the used mathematical operations in the algorithm and since there is no assumption for the sign of the weight, the same approach can be also used for finding the maximum directed spanning tree problem by altering the weight sign. In the following text MST will refer to a *rooted maximum directed spanning tree*.

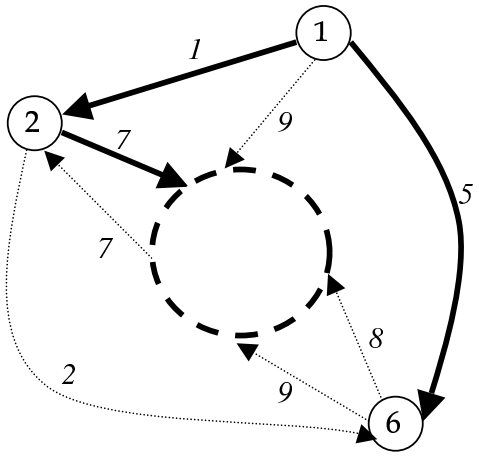


Figure 4.3: MST pseudo-vertex

This MST algorithm will be the core best-tree searching technique to be applied in the techniques and learning paradigms in Chapter 5, Chapter 6 and Chapter 7.

5. Naive Parsing

5.1. The grammar and sentence graphs

Definition: A *grammar graph* $\text{Gram}_L(\mathbf{N}, D)$ is an oriented graph, where **N** is a set of sets N of units of a natural language L (vertices), while D is a set of oriented edges (f, s), where $f \in N$ and $s \in N$, such that f and s form a syntactic dependency as defined within a given consistent linguistic theory on L. The orientation of the edge is determined by the linguistic theory.

Definition: A *complete grammar graph* $\text{Grammar}_L(\mathbf{N}, D)$ is a grammar graph $\text{Gram}_L(\mathbf{N}, D)$, such that $D \neq \varnothing$ and $\forall(f, s) \in$ complement(D) it is not true that (f, s) is allowed within a consistent linguistic theory on L.

The benefit from a complete grammar graph would be significant, mainly because it could represent a "grammatical" space inside of which linguistic problems could be embedded.

A treebank, well-saturated in terms of the linguistic elements of interest, could give an approximation of such aim-specific complete grammar graph. For an analytical dependency tree building based on a morphemic input such graph can be approximated as a collection of all dependencies found in the treebank, which vertices are morphemic units. In such case **N** would consist of one-member sets N, each of which representing a single morphemic tag. (f, s) is an edge if $f = \text{gov}(s)$, where gov(.) is a governor function as defined by the treebank at hand and both $\{f\}$ and $\{s\}$ are members of **N**.

Definition: An *(approximate) treebank graph* $\text{Grammar}_{TB}(N, D, g)$ is an oriented graph, which members of N are obtained by the function g on treebank nodes, and which edges are $(n, m) \in D$, where $n, m \in N$ such that $n = \text{gov}(m)$.

It is not excluded that an approximate treebank graph can be manually enriched by linguistically relevant edges. The PDT treebank graph on morphemic tag vertices has a density[19] of approx. 27% and a most likely coverage with a mean of 99.7%. An expected lower limit of the coverage in PDT with morphemic tag vertices is 99.5% (assuming that there is 1 new dependence for every 10th sentence as observed in the saturation curve Figure 2.2).

Let $\text{ord}_{Snt}(n)$ be a function which output is a surface order of the input element n of sentence Snt.

Definition: A *sentence graph* Snt(N', D', f) is a sentence subgraph of $\text{Grammar}_{TB}(N, D, f)$ of the sentence Snt obtained as follows: $n' = \langle n,i \rangle \in N'$, if $n \in N$ for $i = \text{ord}_{Snt}(n)$, and $(f', s') \in D'$ if $(f, s) \in D$.

Any of the above stated graphs can also be weighted.

---

[19] By a *density* I understand, as it is usually the case, the ratio between $|D|$ (the number of edges of the graph) and $|N|^2 - |N|$ (i.e., the number of edges of a complete graph on the same vertices).

5.2. MST[20] as a dependency tree finder using frequencies

When applied on a weighted sentence graph, MST can be used as a dependency tree finder. Here is one example.

The sentence

```
cmpr9406:001-p8s3    /Z#---    0        <sentence id>
Telefonicky          /Dg---    1        Over phone
jej                  /PPZS4    2        to him
požádal              /VpYS-    3        asked
,                    /Z,---    4        ,
aby                  /J,---    5        that
by                   /Vc-X-    6        would
mu                   /PHZS3    7        him
dokument             /NNIS4    8        document
poslal               /VpYS-    9        send
.                    /Z.---    10       .
```

has the following sentence graph

```
Sntgraph "cmpr9406:001-p8s3" {
    "Z#---.0" -> "Dg---.1" [weight=2];
    "Z#---.0" -> "VpYS-.3" [weight=24];
    "Z#---.0" -> "Z,---.4" [weight=11];
    "Z#---.0" -> "J,---.5" [weight=3];
    "Z#---.0" -> "NNIS4.8" [weight=5];
    "Z#---.0" -> "VpYS-.9" [weight=24];
    "Z#---.0" -> "ZT---.10" [weight=368];
    "Dg---.1" -> "Z,---.4" [weight=4];
    "Dg---.1" -> "J,---.5" [weight=13];
    "VpYS-.3" -> "Dg---.1" [weight=5];
    "VpYS-.3" -> "PPZS4.2" [weight=1];
    "VpYS-.3" -> "Z,---.4" [weight=11];
    "VpYS-.3" -> "J,---.5" [weight=15];
    "VpYS-.3" -> "Vc-X-.6" [weight=5];
    "VpYS-.3" -> "PHZS3.7" [weight=2];
    "VpYS-.3" -> "NNIS4.8" [weight=2];
    "Z,---.4" -> "Dg---.1" [weight=2];
    "Z,---.4" -> "VpYS-.3" [weight=1];
    "Z,---.4" -> "J,---.5" [weight=2];
    "Z,---.4" -> "VpYS-.9" [weight=1];
    "J,---.5" -> "VpYS-.3" [weight=17];
    "J,---.5" -> "Z,---.4" [weight=151];
    "J,---.5" -> "VpYS-.9" [weight=17];
    "NNIS4.8" -> "Z,---.4" [weight=1];
    "NNIS4.8" -> "J,---.5" [weight=3];
    "VpYS-.9" -> "Dg---.1" [weight=5];
    "VpYS-.9" -> "PPZS4.2" [weight=1];
    "VpYS-.9" -> "Z,---.4" [weight=11];
    "VpYS-.9" -> "J,---.5" [weight=15];
    "VpYS-.9" -> "Vc-X-.6" [weight=5];
    "VpYS-.9" -> "PHZS3.7" [weight=2];
    "VpYS-.9" -> "NNIS4.8" [weight=2];
    }
```

[20] As written in Chapter 4, MST stands for rooted maximum directed spanning tree.

obtained from the PDT treebank graph, the vertices of which are the first 5 positions of a morphemic tag. The only modification is done for the Z tags (Z is a punctuation tag and Czech morphemic tags group almost all punctuation signs into a single tag "Z:---"), where the second tag position is replaced by the first letter of the corresponding word form. This change allows one to distinguish between various punctuation signs that was observed to be of benefit for parsing.

Figure 5.1 represents graphically the above sentence graph with the MST tree edges bolded. The weights are frequencies of the dependency as in PDT.

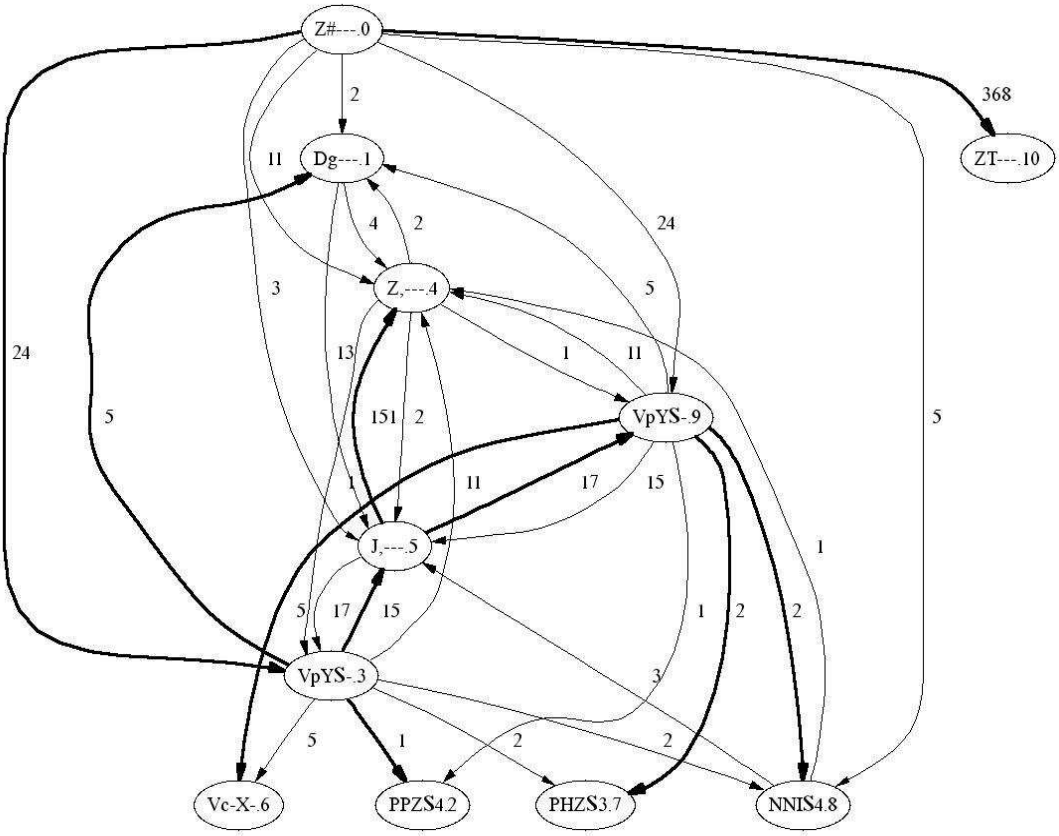


Figure 5.1: An example of a sentence graph

This reveals the first naive parser to be described in the next section.

5.3. The frequency MST parser

The frequency MST parser is obtained as follows:

1. For the input sentence find its weighted *sentence graph* as a subgraph of the weighted PDT treebank graph. The weights are simple frequency records as number of occurrences of the dependencies on the whole PDT, inherited from the (PDT) treebank graph.
2. Modify the weights taken from the PDT treebank graph.
3. Find MST and present it as the parser's output.

Table 5.1 presents the results of this approach: each row represents a different weight modification (step 2), while each column represents a different node value of the PDT treebank graph.

Let *freq* be a frequency of an edge od PDT treebank graph, while let w be the corresponding weight of the sentence graph as a result of step 2.

For the weight modifications:

Freq: $w = freq$, means that no modification was done

Freq/m: $w = freq/m$ means that the weight was made inversely proportional to the dependency length[21] m.

log(Freq): $w = \log(freq)$ means that the weight was transformed using a logarithm of base 10.

step(Freq): $w = \text{step}(freq) = \begin{cases} 0,... freq \leq 10 \\ 1,... 10 < freq \leq 100 \\ 2,... 100 < freq \leq 1000 \\ 3,... freq > 1000 \end{cases}$

For the node definition of the PDT treebank graph:

m2: The first two positions of the morphemic tag with Z tag modified as stated earlier.

m5: The first five positions of the morphemic tag with Z tag modified as stated earlier.

lemma: Lemma values except for the sentence root, which is Z#.

m5l: A combination of m5 and lemma; m5 is taken except for the Z tag modified as stated earlier, for the cases of a verb, conjunction and preposition nodes the node is a concatenation of its m5 and lemma.

Success rate	m2	m5	lemma	m5l
Freq	32.15%	52.40%	46.90%	46.20%
Freq/m	51.50%	**63.27%**	58.65%	54.87%
log(Freq)	31.87%	52.40%	46.90%	46.22%
step(Freq)	18.10%	34.93%	44.28%	37.78%

Table 5.1: Success rates of a frequency-based parser

The results presented in Table 5.1 are obtained on the whole evaluation set of PDT. The percentages are very stable, i.e. when tested on the training data, the most successful frequency MST parser (m5, Freq/m) achieves almost the same result as on the evaluation data, that is 63.96%. The lemma and m5l variants demonstrate lower values because of non-saturated treebank with respect to

[21] As in Chapter 2, by a dependency length I denote the difference of surface order position; neighboring words have a dependency length (or distance) 1, words with a single word between them have a dependency length of 2, etc.

these node values. Opposite to that, m2, although a part of a saturated treebank, oversimplifies the reality - there is a low number of distinct features found in average within a single sentence (often repetition of mutually equal vertex values).

It is of big importance to notice that the effect of distance expressed via the inverse of a distance of the connected vertices causes more than 10% increase of the success rate (almost 20% for m2). The combination of the sentence graph, which is not limited to any distance of the vertices, combined with a inverse distant constraint becomes therefore a significant observation for the future development.

Although for greater distances the inverse of the distance becomes a very small value it is still important, as shown in Table 5.2 calculated for the (Freq/m, m5, cases from Table 5.1); *m* is a distance value, which, in this case, if exceeded causes that the weight are set to 0. From Table 5.1 and Table 5.2 it can be noticed that distances even longer than 15 have non-neglecting influence of 2% on the success rate.

m	Success rate (%)
5	58.46
7	59.07
10	59.94
11 (average distance)	60.23
15	61.25

Table 5.2: Success rate based on sentence length

The almost underlying linear dependence of dependency accuracy on graph density from Figure 5.2 denies a possible claim that the more dense the graphs is the less successful the selection of the correct tree is. Figure 5.3 presents an underlying linear trend with a positive slope between a sentence graph density and a total number of sentence graph edges.

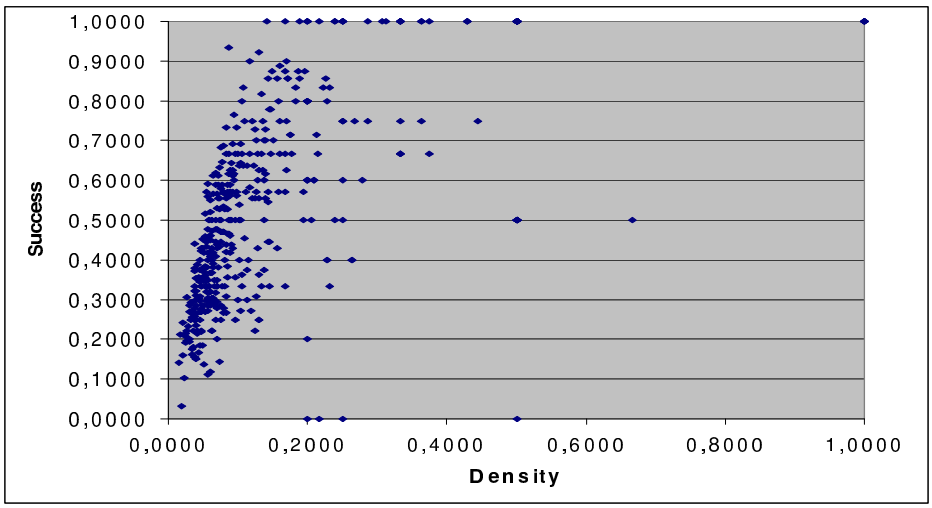


Figure 5.2: Dependency accuracy vs. sentence graph density

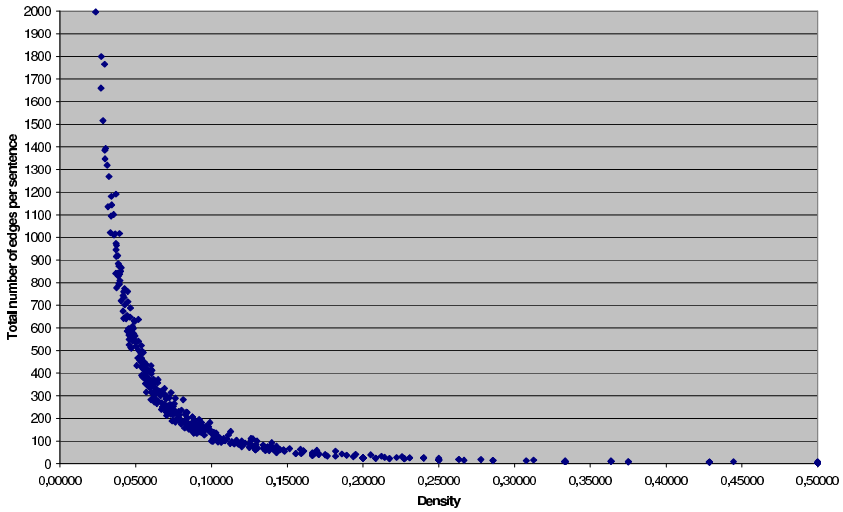


Figure 5.3: Distribution of edges in sentence graphs

The inverse distance was empirically determined as the best one - besides to the above stated weight modifications, square distance inverse and negative exponential distance modifications were also tested but did not show to be as successful as the simple inverse distance.

It is also pleasant to notice that 100% well-parsed sentences were not rare. The following are several such examples:

(a) Free of errors:

`(nd91301:040-p2s1B : l109.am, #50):`
První vánoce oslavili studenti a učitelé na lužaneckém soukromém gymnáziu v Brně.
(First Christmas celebrated students and teachers at the_Lužany's private gymnasium in Brno)

`(lnd91301:038-p2s3 : l109am, #34):`
Po pětadvaceti letech patřila mezi první signatáře Charty 77.
(For twenty-five years (she) belonged among the_first signers (of) Charta 77.)

`(lnd91301:033-p2s1 : l108am, #24):`
Milí přátelé, čtenáři a příznivci.
(Dear friends, readers and favorers.)

`(lnd91301:004-p8s4 : l103am, #4):`
Stal se vůči věroukám pronikavě myslicím skeptikem.
((He) Became *refl.part(became)* against religions a_pungently thinking sceptic.)

`(lnd91301:014-p1s1A : l104am, #49):`
Ukrajina chce chránit svůj trh
(Ukraine wants to_defend its market)

`(lnd91301:030-p2s7 : l108am, #5):`

Tam napadlo lidi vloupat se o vánocich k někomu jinému pouze 137 krát.
(There struck people to_housebreak *refl.particle(housebreak)* at Christmas at someone else's only 137 times.)

(b) Although very rare, unfortunately, it also happens that there are sentences parsed with 0% success:

`(lnd91301:016-p4s2 : l106am, #3):`

Spíš se však údajně opět ztratil.
(Rather *refl.part.(disappeared)* however reputedly again disappeared.)

The results in Table 5.1 obtained without learning determine relatively high parsing baselines.

Values of these baselines, for the case of Freq/m and m5, calculated on sentences with distinct number of verbs are presented in Table 5.3. The dependence on the number of verbs shows that more attention should be paid to the determination of the root (and upper tree levels - a way to approach to partial analysis).

# Verbs	Frequency MST (Freq/m, m5)
0	67.35%
1	70.37%
2	62.02%
3	58.24%
4	56.60%
> 4	55.20%

Table 5.3: Dependency accuracy vs. number of verbs

No increment of success was observed if other local restrictions were made as:

- do not allow edges which cross an odd number of left or right brackets (parenthetical sentence parts)
- do not allow edges which connect an element inside quotes with an element outside of them.

The frequency calculations do not distinguish between left and right dependencies, i.e. relative surface position between father and son vertices. It is expected that in this case the success would improve.

It is interesting to note that if the sentence length is at most 8, for the case of one-verb sentences, the success of the frequency model is the same as the currently best statistical parser for Czech, i.e. 84%. If not limited by the one-verb condition this success is 77%.

Other frequency based experiments with respect to parsing can be found in (Holan 2003, 2004).

5.4. MST as a dependency tree finder using dependency pictures

In the search of suitable rule templates, I wanted to observe the diversity of possible contexts of a dependency within the sentences they occur. The following

experiment has been performed, and that is how our second naive parser was born.

Let $(f, s)_{Snt}$ be a dependency from the sentence Snt. Let $l = |Snt|$ be its sentence length. Let n be the pair $(v, freq)$, where v is the node value and $freq$ is its frequency in a sentence segment. Let $i = \min\{ord_{Snt}(f), ord_{Snt}(s)\}$ and $j = \max\{ord_{Snt}(f), ord_{Snt}(s)\}$. The dependency breaks the sentence Snt into three segments:

- segment *before*: $B_{(Snt,f,s)} = \{n_0, n_1, \cdots, n_{i-1}\}$
- segment *in*: $I_{(Snt,f,s)} = \{n_{i+1}, n_{i+2}, \cdots, n_{j-1}\}$
- segment *after*: $A_{(Snt,f,s)} = \{n_{j+1}, n_{j+2}, \cdots, n_l\}$

For the case when any of these sets within a single sentence is empty a node value of "NONODE" is assigned.

Let

$$B^+_{(f,s)} = \bigcup_{\substack{\forall(f,s)\\ \forall Snt}} B_{(Snt,f,s)},$$

$$I^+_{(f,s)} = \bigcup_{\substack{\forall(f,s)\\ \forall Snt}} I_{(Snt,f,s)} \text{ and}$$

$$A^+_{(f,s)} = \bigcup_{\substack{\forall(f,s)\\ \forall Snt}} A_{(Snt,f,s)}.$$

The general quantifiers are across all PDT sentences from the analytical general training set of PDT.

The triple $\left(B^+_{(f,s)}, I^+_{(f,s)}, A^+_{(f,s)}\right)$ forms a *picture* of the dependency (f, s). In this case the picture will be called a positive one, $Pic^+(f,s)$.

Similarly to the positive picture a negative one $Pic^-(f,s)$ is built, i.e. a picture of all pairs (f, s) for which there is already a positive picture across all sentences, but in the case of the negative picture the *before*, *in* and *after* segments are for the cases when f and s exist within a sentence and it is true that $f \neq gov(s)$.

Examples of picture elements follow. The '+' sign represents a positive picture element, in this case A^+(NNFXX, II---) and A^+(NNFS7, AGFS2), '-' is its negative counterpart. 'r' means that the dependency is a right one, i.e. ord(NNFXX) < ord(II---), while value 'l' would mean, that ord(NNFXX) > ord(II---). 'A' (other possible values are 'I' for *in* and 'B' for *before*) corresponds to the segment *after* as described above. The number attached to the tag values forming the picture element is a frequency of occurrence of the tag within the segment found across PDT.

```
+ NNFXX II--- r A: NNIS2_2 AAFP2_1 Z:---_9 NNFP1_1 NNFP2_2
                   NNNS4_1 PLNS1_1 NNFXX_1 PDNS1_1 RR--4_2
                   Vi-P-_1 VB-S-_1 C=---_3 NNNP1_2 TT---_1
```

```
                    NNFS4_1 NNIS1_2 PDZS3_1 J^---_1 NNNS1_1
                    J,---_2

- NNFXX II--- r A: NNIS2_2 NNNS2_1 AAFP2_1 Z:---_11 NNFP1_1
                    NNFP2_2 NNNS4_1 PLNS1_1 NNFS7_1 Db---_1
                    PDNS1_1 RR--4_2 RR--3_1 Vi-P-_1 VB-S-_3
                    C=---_2 NNNP1_2 AAFS3_1 NNFS4_1 NNIS1_2
                    PDZS3_1 RR--6_1 J^---_4 NNIS4_1 NNMS1_4
                    NNFS6_1 J,---_3 NNFS3_1

+ NNFS7 AGFS2 r B: RR--2_1 VB-S-_1 Z:---_1 NNFP2_1 Z#---_2
                    J^---_1 Db---_2 NNMS7_1 PDNS1_1 AAFS2_2
                    NNFS2_2

- NNFS7 AGFS2 r B: NNFP2_1 VpTP-_1 VsTP-_1 PDYS1_1 NNFXX_1
                    NNIS7_1 II---_1 NNIP1_1 VsQW-_1 RR--4_2
                    NNIS6_4 NNIXX_1 RR--3_1 AAFS2_5 NNFS2_7
                    VB-S-_3 C=---_3 AAFS6_1 NNFS4_1 Z#---_8
                    NNIS1_4 J^---_2 NNMP2_1 AAIS1_1 AAIS6_1
                    RR--7_1 NNFS6_2 NNIP4_1 NNMS2_1 NNIS3_1
                    NNIS2_1 NNNS2_1 Z:---_11 AAXXX_1 P7-X4_3
                    VpQW-_1 Vf---_1 Db---_3 ClXP2_1 AGIS7_1
                    NNNS6_3 TT---_2 AAMP2_1 RR--6_9 AUIS6_1
                    NNMS1_4 NNFS1_1
```

Once the pictures are formed, it is possible to approach towards a picture-based dependency weight determination. A simple solution towards the dependency weight determination is to perform the following *frequency-based intersection* of the corresponding segments of a positive and a negative picture. Assuming that a non existent elements has a frequency of zero, the frequency based intersection of a segment is defined as $\sum_{\forall n \in segment} (freq^{+}(n) - freq^{-}(n))$, where $freq^{+}(n)$ is the frequency of node n as found in the positive picture, and $freq^{-}(n)$ is the frequency of node n as found in the negative picture.

This type of picture evaluations is shown in the following example with two short sentences, where the second one was deliberately chosen to demonstrate a situation when the pictures did not add extra knowledge. This is more likely to happen for short sentences.

Třikrát/Cv---	rychlejší/AAFS1	než/J,---	slovo/NNNS1
Three-times	faster	than	a word

```
1 AAFS1 Z#--- r S=-0.167 D=2  B=0 I=0 A=-0.5
1 Cv--- AAFS1 l S=-0.33  D=1  B=0 I=-1 A=0
1 J,--- AAFS1 r S=1      D=1  B=0 I=0 A=0
1 NNNS1 J,--- r S=1      D=1  B=0 I=0 A=0
```

Pomocí/NNFS7 ECM/NNIXX	může/VB-S-	být/Vf---	systém/NNIS1
A help ECM	can	be	the system

```
1 VB-S- Z#--- r S=0.5    D=2  B=0 I=0 A=0
0 NNFS7 VB-S- l S=1      D=1  B=0 I=0 A=0
1 NNFS7 Vf--- l S=0.5    D=2  B=0 I=0 A=0
0 Vf--- NNFS7 r S=0.5    D=2  B=0 I=0 A=0
```

```
0 VB-S- Vf--- l S=1       D=1  B=0 I=0 A=0
1 Vf--- VB-S- r S=1       D=1  B=0 I=0 A=0
1 NNIS1 VB-S- r S=0.5     D=2  B=0 I=0 A=0
0 NNIS1 Vf--- r S=1       D=1  B=0 I=0 A=0
0 NNIS1 NNIXX l S=1       D=1  B=0 I=0 A=0
1 NNIXX NNIS1 r S=1       D=1  B=0 I=0 A=0
```

The first number in each row represents whether the dependency is a PDT tree dependency or not. The following two elements are the father and the son node values. 'r' or 'l', as earlier in the text, distinguish between a left and a right dependency. This time, the values of B, I and A, correspond to the frequency based intersection of the *before*, *in* and *after* segments divided by the segment length (number of nodes in the segment in the analyzed sentence; if zero, this length is set to 1). D is a distance of the dependency elements, expressed in number of elements. S is a value, which combines everything into a single weight value.

With respect to the notation from the sentence examples

$$S = \alpha_1 B + \alpha_2 I + \alpha_3 A + \alpha_4 \delta(B, I, A), \text{ where}$$

$$\delta(B, I, A) = \begin{cases} \frac{1}{D^m}, \ldots B = 0, I = 0, A = 0 \\ 1, \ldots (\alpha_1 B + \alpha_2 I + \alpha_3 A \geq 0), \neg(B = 0, I = 0, A = 0), \text{ where } m \geq 1. \\ 0, \ldots otherwise \end{cases}$$

In the following experiments S becomes a dependency weight of a sentence graph. Afterwards, the dependency tree is found by applying the MST algorithm. Table 5.4 presents results of this experiment for various values of the α coefficients.

Parameter values	**Success rate (%)**
$\alpha_1 = 0, \alpha_2 = 0, \alpha_3 = 1, \alpha_4 = 0, m = 1$	52.95
$\alpha_1 = 1/3, \alpha_2 = 1/3, \alpha_3 = 1/3, \alpha_4 = 1, m = 1$	**68.34**
$\alpha_1 = 1/3, \alpha_2 = 1/3, \alpha_3 = 1/3, \alpha_4 = 1, m = 2$	68.30
$\alpha_1 = 1/3, \alpha_2 = 1/3, \alpha_3 = 1/3, \alpha_4 = 0$	52.95
$\alpha_1 = 1, \alpha_2 = 0, \alpha_3 = 0, \alpha_4 = 0$	51.41
$\alpha_1 = 0, \alpha_2 = 1, \alpha_3 = 0, \alpha_4 = 1, m = 1$	53.99[22]
$\alpha_1 = 0, \alpha_2 = 1, \alpha_3 = 0, \alpha_4 = 0$	58.06
$\alpha_1 = 0.2, \alpha_2 = 0.5, \alpha_3 = 0.3, \alpha_4 = 1, m = 1$	68.31

Table 5.4: Dependency accuracy of a picture-based parser

[22] The achieved 53.99% on the *in* segment are lower than one would expect.

For there is no automatic learning adjustment of the parameters, the results are flattering. Of course, they are far below the best current parser for Czech (84%). Nevertheless, they are cheap to obtain and their performance is better than, e.g. pure probabilistic methods as originally applied on Czech.

With respect to the creation of rule templates needed for automatic rule generation, this analysis brings the following results:

- Most attention should be paid to features between the dependency elements.
- Distance between the dependency elements could be employed, especially for the situations when the parser cannot decide which dependency to prefer.
- There is non-neglecting information also in the *before* and *after* segments of the sentence, where the *after* segment is slightly more contributive than the *before* one.
- For the first time, it is interesting to note the negative effect on the distance for the *in* segment. With a special attention to linguistic relevance of features, this indicates a possibility that most attention should be paid to features defined with elements/values located between the dependency elements.
- It is pleasant to notice that the pictures are almost insensitive to sentence length. This is an important difference from other approaches.

The insensitivity to sentence length of the picture parser compared to the high success rate of frequency-based one for short sentences, is a key how to combine them. A sensible combination would yield at least 70% success rate.

Undoubtedly, it would be better to have a higher than a 70%-level success. Nevertheless, the results of this kind are not as bad for, e.g. information extraction systems, especially for parsing of shorter sentences or questions and/or sentences of common type where additional 'sharpening' of the parser can pop up the success rate.

For parser evaluations, the frequency based MST dependency tree finder, offers, mainly thanks to the MST algorithm, a new and relatively high baseline.

Thanks to the availability of the MST algorithm, in the following chapter, the logic of RBA will be applied for modification of sentence graph weights in order to obtain such MST, that will as close as possible to the true, PDT tree.

5.5. The percentages of naive parsing

Table 5.5 joins the results from this chapter into a single table. The `anal_general_train` are PDT general train data for the analytical PDT layer; the `anal_general_etest` are PDT general evaluation test data for the analytical PDT layer. `anal_general_train` consists of 1583 files, while the `anal_general_etest` consist of 161 files. Each file consists of approx. 50 sentences.

Algorithm	Size and identification of train data	Size and identification of test data	Dependency accuracy	Note
Frequency-based MST parser, Section 5.3	anal_general_train	anal_general_etest	63.27%	Freq/m and m5, as in Table 5.1.
	the same as above	on the train data	63.96%	Freq/m and m5, as in Table 5.1.
	the same as above	anal_general_etest	58.46%	Freq/m and m5, and not considering dependency length bigger than 5.
	the same as above	*the same as above*	59.07%	Freq/m and m5, and not considering dependency length bigger than 7.
	the same as above	*the same as above*	60.23%	Freq/m and m5, and not considering dependency length bigger than 11, which is average dependency length.
	the same as above	*the same as above*	61.25%	Freq/m and m5, and not considering dependency length bigger than 15.
	the same as above	All sentences from anal_general_etest with no verbs	67.35%	Freq/m and m5.
	the same as above	All sentences from anal_general_etest with 1 verb	70.37%	Freq/m and m5.
	the same as above	All sentences from anal_general_etest with 2 verbs	62.02%	Freq/m and m5.
	the same as above	All sentences from anal_general_etest with more than 4 verbs	55.20%	Freq/m and m5.
	the same as above	All sentences from anal_general_etest with one verb and at most 8 tokens	84%	Freq/m and m5.
	the same as above	All sentences from anal_general_etest with at most 8 tokens	77%	Freq/m and m5.
the table continues ...				

Picture-based MST parser, Section 5.4 *All pictures consist of m5 type of nodes (the first 5 positions of the morphemic tag), and no lexical units.*	`anal_ general_ train`	on the train data[23]	68.34%	All segments and inverse distance.
	the same as above	*the same as above*	52.95%	All segments and without distance.
	the same as above	*the same as above*	52.95%	Only the *after* segment with inverse distance.
	the same as above	*the same as above*	51.41%	Only the *before* segment without distance.
	the same as above	*the same as above*	53.99%	Only the *in* segment with distance.
	the same as above	*the same as above*	58.06%	Only the *in* segment without distance.
Combination of the above two naive parsers	*the same as above*	*the same as above*	at least 70%	Freq/m and m5, and all segments and inverse distance

Table 5.5: The percentages of naive parsing

[23] The aim of the experiments with pictures was to reveal characteristics of the sentence elements with respect to a syntactic dependency. Therefore, the character of the experiment did not require testing on test data. The results obtained did not depend on sentence length (these experiments are not included in the table).

6. À la RBA

After many trials with traditional type modifications of the originally phrase grammar rule-based approach for parsing (Chapter 1 and Chapter 3), I was curious to find out the possibilities of this paradigm when applied on a sentence graph. The high frequency based success from Chapter 5 benefits from eliminating direct dependence on surface word ordering. The aim has also been to check the advantages of such initial tree structure that manages to group "relevant" sentence nodes. In my previous experiments with RBA the free word ordering of Czech, the non-projective constructions, the dependencies related to nodes to the right and other features caused that the learning process has not been successful enough.

Our aim in this chapter will be to examine whether the RBA paradigm can be applied for sentence graph weight modifications. Although not purely RBA paradigm, I will use this naming since the algorithmic scheme as presented in Figure1.1 and Figure 1.2 remains the same.

The initial structure is a sentence graph with normalized frequencies with values from the interval [0, M], where M is the maximum allowed dependency weight. The nodes are morphemic tags (their first five positions). For the success rate of such initial structure see Table 5.1.

A rule modifies the edge weight by incrementing or decrementing the edge weight by 1.

The aim is to use MST to find the best dependency tree.

The rules are generated in a greedy way, one by one, finding the best rule of each iteration.

The learning stops as soon as there is no further improvement, or if the learning improvement is less than a threshold value ε. Using a threshold value is recommended in order not to over-generalize.

6.1. Learning improvement

The learning improvement of a sentence (sentence graph) is calculated based on the following two values:

$$A = \frac{\sum m}{\frac{MST-tree}{M(N-1)}}, \text{ and}$$

$$B = \frac{\sum m}{\frac{complement(PDT-tree)}{M(D-N+1)}}, \text{ where}$$

m is the dependency weight, M its maximal value, N is the number of vertices in the sentence graph, D is the number of edges of the sentence graph, and the complement is found with respect to the sentence graph.

Our aim would be to improve the *graph condition*, i.e. intuitively but not precisely speaking, to increase A or decrease B. What is meant by improving the graph condition is stated in the following text.

Let A_t is the value of A in iteration t; analogously for B_t.

Instead of the classical error function, $C_t = A_t - B_t$ determines an improvement for iteration t, under the condition that the following statement is true:

$$((A_t \geq A_{t-1}) or (B_t \leq B_{t-1})) and ((A_t - B_t) > (A_{t-1} - B_{t-1}))$$

The value $A_t - B_t$ is usually small and a suitable threshold value ε would be of the order 10^{-8}.

It is also possible to use the classical error function, i.e. a percentage of correct dependencies (E_t). This is not recommended since the modifications of a single rule are very small and it frequently happens that several rules are needed before the MST algorithm outputs a different tree. This was observed from the learning process.

An alternative would be to characterize the learning improvement by the sum $E_t + C_t$, which is the way the improvement was characterized in the experiments.

It is time now to discuss the rule templates.

6.2. The rules

The aim of the rules is to take the surface structure of the sentence and transform it into a simple operation of dependency weight modification, which will contribute towards a better MST performance over the sentence graph.

There are 5 types of rule templates tested:

1. The Simple rule
2. The Concord rule
3. The Distance rule
4. The Between rule
5. The Not-Between rule

Every instance of a rule can be positive or negative, i.e. add or subtract a value from a dependency weight.

All rule instances can be extracted automatically from a treebank.

6.2.1. The Simple rule

The Simple rule is of the following form

```
<father vertex value> <son vertex value> <S type>
```

The `<S type>` are two values: 'l' and 'r' for left and right dependency type. A left dependency is such that ord(`<father vertex>`) > ord(`<son vertex>`); otherwise the dependency is right.

6.2.2. The Concord rule

The Concord rule is of the following form

```
<father vertex value> <son vertex value> <C type>
```

The `<C type>` has three possible values: 3 for gender, 4 for number and 5 for case.

This rule tests concord (agreement) in terms of equal tag position values between the dependency elements.

6.2.3. The Distance rule

This rule was added in order to operate with the distance values. Its form is

```
<father vertex value> <son vertex value> <D type>
```

The `<D type>` is a number representing a distance between the dependency elements. The rule simply tests the surface distance of the two vertices without distinction for left and right dependency orientation.

This rule was added on the basis of the observations from Chapter 5.

6.2.4. The Between rule

The Between rule has the following form

```
<father vertex value> <son vertex value> <B vertex value>
```

This rule is true if the `vertex value` from `<B vertex value>` is present on the sentence surface between the father and the son vertex values.

Similarly as for the *in* sector of the positive pictures from Chapter 5 the rule instances are generated using a treebank.

6.2.5. The Not-Between rule

The Not-Between rule has the following form

```
<father vertex value> <son vertex value> <N vertex value>
```

This rule is true if the `vertex value` from `<N vertex value>` is not present between the father and the son vertex values on the sentence surface form.

Similarly as for the *in* sector of the negative pictures from Chapter 5 the rule instances are generated using a treebank.

6.2.6. Other rules

If one is able to create automatically rule instances of a rule template and their application is not dependent on existence of treebank trees, it would be possible then to use such template in the learning process. There are no limitations on whether the rule templates are manually crafted based on a linguistic intuition or automatically deduced based on a treebank. In this phase, the only limitation is that the rules have to operate on the surface sentence nodes.

6.3. Examples of several rule instances

Some rule instances, taken randomly from the rules file, as used by the `graphrule` program, are listed. Each of them can have its positive and negative variant.

```
RR--7 AANS7 Sr
RR--7 AANS7 C5
Db--- J,--- Sl
NNFS2 NNFP2 Sr
NNFS2 NNFP2 C5
NNFS2 NNFP2 C3
NNIS4 NNFS2 Sr
NNIS4 NNFS2 C4
TT--- Db--- Sl
RR--7 NNNS7 BAANS7
```

The first two members in each row are the first five positions of the morphemic tag, while the third member specifies the type according to its first character: S for simple, C for concord, B for between, N for not between, D for distance. The other symbols in the third member are as previously described in the rule descriptions.

The following is a list of the first 20 best rules found by the `graphrule`[24] program.

```
1  VB-P- NNIS4 Sr +1
2  NNMS1 NNMS1 Sl +1
3  J,--- VB-S- Sr +1
4  J,--- VB-S- Sr +1
5  J,--- VB-S- Sr +1
6  J,--- VB-S- Sr +1
7  J^--- VB-S- Sr +1
8  VB-S- Z,--- Sr +1
9  Dg--- Db--- Sl +1
10 J,--- VB-P- Sr +1
11 VpMP- VB-P- Sl +1
12 Dg--- Db--- Sl +1
13 J^--- VB-S- Sl +1
14 J,--- VB-P- Sr +1
15 VB-S- J^--- Sr +1
16 J,--- Z,--- Sr +1
17 VpMP- VB-P- Sl +1
18 VpMP- VB-P- Sl +1
19 NNFS2 AAFS2 Sl +1
20 NNFXX C=--- Sr +1
```

Their description is the same as of the rule instances; the +1 value means that the weight value is incremented.

---

[24] This and other programs, programmed by the author of this work, are available (for academic purposes) upon request to the author.

6.4. Observations, characteristics and results

The main loop of the `graphrule` algorithm is given below.

```
while there is a learning improvement
        for each Rule from rule instances list
                for each Sentence from all Train Set sentences
                    success = test Rule on Sentence
                end
        end
        select Best Rule, i.e. the rule which has the best performance
                            over the TrainSet
        apply Best Rule  on Train Set
end
```

For a train set of 1000 sentences which contains approx. 83,000 sentence graph dependencies, there are in average about 200,000 rule instances generated.

In the first run `graphrule` indexes the train set with corresponding rule instance such that for each sentence only links to rule instances which can be applied are recorded. Such optimization step significantly improves the algorithm speed. Unfortunately, the time complexity is still high - the training process selects a single best rule in approx. 12 hours on a 2GHz processor system with 1GB of internal memory. This fact influences the size and extend of the experiment testing.

The number of best rules needed is higher than in the classical RBA approach, which is due to the fact that the rules cause small step improvements. A bigger number of training sentences this time influence positively the learning phase.

To be able to draw conclusions on the success of this approach several small experiments (100 sentences) were performed. Before approaching to a further optimization of the complexity of the `graphrule` program, I wanted to find out its prospects first.

After a relatively high number (hundreds) of best rules is found, the success of the learning process of the 100 sentences sets climbs up to 82%, in exceptional cases also 87%.

These results were shown to be non-stable - once the training set size increases the success rate of the learning process decreases.

With respect to the time complexity, I managed to run an experiment with 400 sentences. The registered improvement achieved on a randomly selected test set was up to 3% above the frequency-based experiments described in Chapter 5.

Tests were also done using different weight modification values, starting with a larger value, a multiple of the +1 and -1 of the rules, and decreasing it during the learning process. Tests with bigger steps (larger than +/-1 weight modification values) were experimentally verified as safe and such that speed up the learning process. The weight modification values were similar to addition of a moment for the process of learning.

Although I name this approach with the terminology of RBA, as designed it does not correspond completely to this paradigm. Namely, in our case, the best rules list is not sensitive to its order since a modification of a weigh of a sentence graph dependency is not conditioned by prior such modifications. This is the reason why:

- the performed experiments are mainly suitable for rule templates testing, and
- the performed experiments could become an input for a second phase with a main stress on tree modifications within a 'dependency space' approximately represented by the sentence graph.

The 3% success increase is a small one. Together it does not outperform the picture-based approach. Hence, this is another confirmation (along with those in Chapter 3) that the RBA approach accompanied by simple additives is not suitable for dependency parsing of Czech. Probably, a last experiment could be performed, presented in Section 6.5. This is the last trial on RBA-like modifications and as it can be noticed in Chapter 7, the perceptron-based approach offers better success rate and a big space for its further improvement.

6.5. Definition of a second phase - tree modifications

Defined using PDT trees, but applied for the learning process on MST trees (trees produces by the MST algorithm on the sentence graph), I present two rule templates which can be implemented. Figure 6.1 will be used to explain rule changes; the bold solid lines mean that the dependency is in both PDT tree and MST tree, the bold dotted line mean that the dependency is only a MST one,

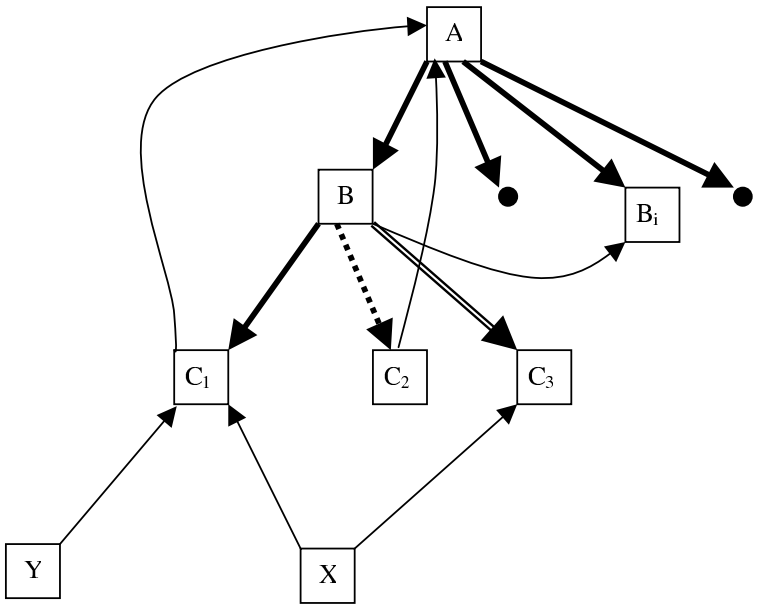


Figure 6.1: A fragment of a sentence graph

while the double stroke line means that the dependency is only a PDT one; the other are the rest of the sentence graph dependencies. Let *SW* represent the total weight of the sentence graph, a sum of weights of all edges. A choice was made that any transformation should keep *SW* constant[25].

6.5.1. Preceding dependency

The rule has the following form

```
<weight modifier> <current dep.> <prec. dep.>
```

The rule modifies the weight of a dependency `<current dep.>` by `<weight modifier>` if a preceding MST dependency is `<prec. dep.>`.

Let us assume the situation from Figure 6.1. An instance of a preceding dependency rule could be:

```
+1 B C1 A
```

where the current dependency is (B, C_1) and the preceding dependency is (A, B); the weight modifier is addition of the value $v = 1$.

If applied on a situation as in Figure 6.1 such rule would modify the weight of (B, C_1) by adding 1, at the same time modify the weights of all other input edges of C_1 by $(-1)v/n$, where $n = \deg_{in}(C_1) - 1$, in this case, the rule will decrease the weight of (X, C_1) and the weight of (Y, C_1) by 1/2.

6.5.2. Brother dependency

Similarly to the previous rule, the brother dependency rule templates support an existence of a dependency based on existence of its brother node.

The rule has the following form

```
<weight modifier> <current dep.> <brother dep.>
```

Modify the weight of a dependency `<current dep.>` by `<weight modifier>` if its brother MST dependency is `<brother dep.>`.

Let us assume the situation from Figure 6.1. An instance of a preceding dependency rule could be:

```
+1 B C1 C2
```

where the current dependency is (B, C_1) and the brother dependency is (B, C_2) ; the weight modifier is addition of a value $v = 1$.

If applied on a situation as in Figure 6.1 such rule would modify the weight of (B, C_1) by adding 1, and at the same time modify the weights of all other output edges of B by $(-1)v/m$, where $m = \deg_{out}(B) - \deg_{MST\text{-}out}(B)$, in this case, the rules will decrease the weight of (B, C_3) and the weight of (B, B_i) by 1/2.

For the MST tree changes, in this second phase it is not possible to index the train set effectively in advance by the rule instances to test.

---

[25] Applying such criterion in the previous phase (as explained in the previous section) means that the ordering of the found best rules matter.

All rule instances can be created automatically from the treebank.

Addition of such tree dependent rules would not be so straightforward for the perceptron-based approach presented in Chapter 7. From this point of view, is to be applied, such types of rules are more suitable for the approach described in this chapter.

6.6. A comment

The ideas presented in this chapter have been my most strongly motivating ones during my studies of RBA and its application to a dependency tree parser. These ideas gave birth to the sentence graphs and to the application of the MST on them. Unfortunately, the RBA approach did not yield a satisfactory level of success.

I have tried to verify proposed solutions through a variety of small experiments. If still one would like to experiment with the RBA type of approaches, this chapter has offered one of the few possible directions. I have tried here to present their core; many modifications are possible, and hopefully some of them, as e.g. the second phase of Section 6.5 could be a subject of study in the near future.

The following chapter will present a method capable of dealing efficiently with large number of features (rules).

6.7. The percentages of à la RBA

The long time needed for training did not allow for detailed testing on larger train sets neither to experiment efficiently with various rule templates. For the aim has been only to verify possible potentials of this approach the training and testing were performed on parts of `anal_general_train` data from PDT. These parts were obtained from `l10*am` (total 9 files) and `c10*am` (total 9 files) files. Each PDT file consists of approx. 50 sentences. Sets of sentences from `c10*am` for training and from `l10*am` for testing were randomly selected - that is the way the number of sentences in Table 6.1 was taken; I do not include there small tests (on less than 50 sentences) or repeated tests. Table 6.1 summarizes the major tests performed in this chapter. If not stated otherwise the sentence graphs were on m5 type of nodes.

Algorithm	Size and identification of train data	Size and identification of test data	Dependency accuracy	Note
À la RBA parsing, Chapter 6	-	-	63%	Initial structure dependency accuracy.
	100 sentences	on the train data	82%-87%	Interval of dependency accuracy on several cross-validation experiments.
	400 sentences	100 sentences	66%	
	the same as above	on the train data	69%	
	`c101.am` file	on the train data	lower than above[26]	On m5l type of nodes.

Table 6.1: The percentages of à la RBA

[26] The improvement has been better, but in the case of m5l nodes the initial dependency accuracy is lower, therefore the whole performance is worse.

7. The Perceptron-Based Approach for Building of a Dependency Tree

Besides Brill's works on RBA (Brill 1993a,b,c), another paper, this time of Michael Collins, has strongly influenced my work: "Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithm" (Collins 2002). In its abstract M. Collins states: "We describe new algorithms for training tagging models, as an alternative to maximum-entropy models of conditional random fields. The algorithms rely on Viterbi decoding of training examples, combined with simple additive updates. ...."

In this chapter I will explain a modification of the perceptron algorithm in order to use it for dependency parsing. The algorithm will rely on MST decoding of the training sentence graphs (training examples), combined with simple additive updates.

In the sequel, firstly, I will explain the main ideas of the perceptron algorithm as applied by Collins on the tagging problem, and secondly, I will state the modifications of the perceptron algorithm for dependency parsing. Thirdly, I will demonstrate the application of the new perceptron-based dependency tree parser. To ease reading, I will try to use, where appropriate, the same notation as in (Collins 2002).

7.1. The perceptron algorithm for tagging

Let $w_{[1:n]}$ be a shorthand for a sequence of words $[w_1, w_2, \cdots, w_n]$, and $t_{[1:n]}$ a shorthand for a sequence of tags $[t_1, t_2, \cdots, t_n]$. Let the training set consist of n tagged sentences, the i'th sentence being of length n_i. Let the training set examples are $\left(w^i_{[1:n_i]}, t^i_{[1:n_i]} \right)$ for i=1...n.

Then the training perceptron algorithm (for tagging) is as follows:

- Choose a parameter T defining the number of iterations over the training set. This parameter is chosen experimentally on the basis of the development set, or alternatively on the basis of the observed saturation of the γ parameter values (see below).
- Initially set all α_{f_k} parameters to 0. For each feature type f_k there is a separate α_{f_k}. Let m be the number of all available features.
- for t = 1 to T
  for i = 1 to n
  Use the Viterbi algorithm to find the best tagged sequence of $w^i_{[1:n_i]}$ under the current parameter settings. Let this sequence be called $z_{[1:n_i]}$.
  for k = 1 to m

c_1 = sum of all outputs of feature f_k in $t_{[1:n_i]}$, and

c_2 = sum of all outputs of feature f_k in $z_{[1:n_i]}$.

Modify the corresponding α_{f_k} parameter as follows:

$\alpha_{f_k} := \alpha_{f_k} + c_1 - c_2$

end for *k*

end for *i*

end for *t*

The γ parameters, mentioned before, are averaged α parameters. They can be obtained easily during the learning process along with the α parameters, as follows:

$$\gamma_{f_k} = \sum_{\substack{t=1...T \\ i=1...n}} \frac{\alpha_{f_k}^{t,i}}{nT},$$

where $\alpha_{f_k}^{t,i}$ is the value of α_{f_k} in the *t*'th iteration of the *i*'th training example.

In the Collins' experiments and as well as in the experiments presented in this chapter, the averaged parameters perform significantly better. The paper (Collins 2003) also presents a justification of the averaging method. The same paper gives proofs to theoretically important aspects of the perceptron algorithm with respect to the upper limit of the number of mistakes for a given training set (see also (Mitchell 1997)), proving its robustness with respect to the train set.

For tagging Collins uses two types of features:

- tag trigrams,
- lexical unit and its tag.

The following sentence from the training set[27]

the/D man/N saw/V the/D dog/N

contains the following tag trigram features

DNV NVD VDN

and the following lexical unit and its tag features

the->D, man->N, saw->V, dog->N.

Let the following be the output of the Viterbi algorithm in some of the stages of the perceptron algorithm (the wrong tag assignment is bolded):

the/D man/N saw/**N** the/D dog/N.

Then, the parameter modification will:

add 1 to: DNV, NVD, VDN

substract 1 from: DNN, NND, NDN, saw->N.

[27] The examples on tagging are taken from (Collins 2003).

7.2. Formalizing the features

Different types of problems may have different feature-vector representations.

7.2.1. Tagging

For tagging, a feature-vector representation ϕ:H × T → $\mathbf{R}^m$ is a function ϕ that maps a history-tag pair to a m-dimensional feature vector. For the tagging experiment, e.g. in maximum entropy model a history h_i from H is a 4-tuple $\langle t_{-1}, t_{-2}, w_{[1:n]}, i \rangle$. In general, the history can be any allowable input element.

Each component $\phi_k(h,t)$ for k = 1...m could be an arbitrary function of (h, t). It is common that each feature is an indicator function (Ratnaparkhi 1996) as, e.g. for tagging:

$$\phi_{1000}(h,t) = \begin{cases} 1,...if\ \ current\ \ word\ \ w_i = the\ \ and\ \ t = DT \\ 0,...otherwise \end{cases}$$, or

$$\phi_{1001}(h,t) = \begin{cases} 1,...if\ \ current\ \ tag\ \ trigram\ \ is\ \ D,N,V \\ 0,...otherwise \end{cases}$$

7.2.2. Dependency parsing

To be able to define similar features for parsing the feature vector representation needs to be defined on dependencies instead of words. In such case ϕ:**H** × **D** → $\mathbf{R}^m$, where the history **H** contains all sentence elements and all sentence dependencies; **D** is a set of dependencies. More formally we could write that **H** members are of the type $\langle w_{[1:n]}, d_{[1:s]}, i \rangle$ where $d_{[1:s]}$ stands as a shorthand for all dependencies of a sentence graph Snt(N', D', g) ordered as father, son, by their surface sentence order. This time, the index i indexes the dependencies. The governor relation we will need is included in $d_{[1:s]}$. Without loss of generality one may assume that g outputs the morphemic tags of the sentence elements. One may then define features as

$$\phi_{123500}(h,d) = \begin{cases} 1,...if\ \ current\ \ dependency\ \ is\ \ (N,A) \\ 0,...otherwise \end{cases}$$

or

$$\phi_{58000}(h,d) = \begin{cases} 1,...if\ \ current\ \ dependency\ \ is\ \ (V,R)\ \ and\ \ N\ \ is\ \ between\ \ on\ \ surface \\ 0,...otherwise \end{cases}$$

7.2.3. Global representations

The ϕ functions are called local function representations. A global representation is,

for tagging

$$f_k\left(w_{[1:n]}, t_{[1:n]}\right) = \sum_{i=1}^{n} \phi_k(h_i, t_i), \text{ or}$$

$$f_k\left(w_{[1:n]}, d_{[1:s]}\right) = \sum_{i=1}^{s} \phi_k(h_i, d_i)$$

for parsing.

Each global feature f_k is a simple sum of local features ϕ_k over sentence elements. If the local features are indicator functions, as it will be in our case, then the global features will typically be simple counts.

7.3. Score

Given a sequence of words $w_{[1:n]}$ and a sequence of tags $t_{[1:n]}$ a score of a tagged sequence is

$$\sum_{i=1}^{n}\sum_{k=1}^{m} \alpha_k \phi_k(h_i, t_i) = \sum_{k=1}^{m} \alpha_k f_k\left(w_{[1:n]}, t_{[1:n]}\right).$$

Given a sequence of words $w_{[1:n]}$ and a sequence of dependencies $d_{[1:s]}$ a score of a tagged sequence is

$$\sum_{i=1}^{s}\sum_{k=1}^{m} \alpha_k \phi_k(h_i, d_i) = \sum_{k=1}^{m} \alpha_k f_k\left(w_{[1:n]}, d_{[1:s]}\right).$$

7.4. Perceptron-based dependency parsing algorithm

The following is the perceptron-based algorithm for dependency parsing.

7.4.1. Learning

Input:

- A training set of sentence graphs with treebank info, Snt(N', D', g) ≡ $\left(w^i_{[1:n_i]}, d^i_{[1:s_i]}\right)$ for i=1...n. Let $\tau^i_{[1:n_i-1]}$ be the dependency tree of the sentence nodes $w^i_{[1:n_i]}$. Snt(N', D', g) is formed based on a treebank grammar graph Grammar$_{TB}$(N, D, g), in our case a PDT grammar graph. g is a user defined function of sentence nodes, most commonly giving a morphemic tag at its output.
- A parameter T determines number of iterations over the training set.
- One also needs a set of local features. More on the local features and their global representation f as applied on Czech analytical tree building is discussed later in the text.

Initialization:

- Set parameter vector $\bar{\alpha} = 0$.

Algorithm:

For t = 1 to T

For i =1 to n

Use the MST algorithm to find the output of the i'th training sentence graph Snt_i(N', D', g) with the current parameter settings, i.e.

$$\zeta_{[1:n_i-1]} = MST\left(\mathbf{Snt}_i(\mathbf{N'},\mathbf{D'},g): weight = \sum_{k=1}^{m} \alpha_k f_k\left(w_{[1:n]}, d_{[1:s]}\right)\right)$$
$$\forall(f,s)$$

where the i'th sentence graph is weighted by the sum of the components of the scalar product of the global feature vector and the parameter vector $\bar{\alpha}$, i.e. the score of the rooted directed maximal spanning tree of the sentence graph. $(f,s) \in \mathbf{D'}$.

Let $\tau^i_{[1:n_i-1]}$ be the true (PDT) tree of the i'th sentence.

If $\tau^i_{[1:n_i-1]} \neq \zeta_{[1:n_i-1]}$ then update the parameters of the parameter vector as follows:

$$\alpha_k := \alpha_k + f_k\left(w^i_{[1:n_i-1]}, \tau^i_{[1:n_i-1]}\right) - f_k\left(w^i_{[1:n_i-1]}, \zeta^i_{[1:n_i-1]}\right)$$

end for i

end for t

Output:

- Parameter vector $\bar{\alpha}$ (and/or parameter vector $\bar{\gamma}$).

The score mentioned in the algorithm is defined in the similar way as the score of the tagged sequence where the sum loop is over the spanning tree.

7.4.2. Application

Once having learned the coefficients, the application is straightforward:

Input:

Sentence and its sentence graph with zero weights and the parameter vector $\bar{\alpha}$ (or the parameter vector $\bar{\gamma}$). As mentioned earlier in the text, selecting the parameter vector $\bar{\gamma}$ significantly increases the success rate. In our experiments aprox. 5% higher success rate was obtained if the $\bar{\gamma}$ vector was used instead of the $\bar{\alpha}$ one.

Algorithm:

Apply the MST algorithm on the sentence graph where each dependency is weighted by its score according to $\bar{\gamma}$ (or $\bar{\alpha}$).

Output:

Dependency tree of the input sentence.

7.5. Feature types

The feature types are very similar to those used in the previous chapter. The similarity can be shown on the following example of some randomly selected features.

```
Db--- PLNS1 Sr
Db--- PLNS1 BNOTAG
Db--- PLNS1 L3
Db--- PLNS1 NNOTAG
NNIXX AAMP2 Sl
NNIXX AAMP2 BNOTAG
NNIXX AAMP2 L3
NNIXX AAMP2 NNOTAG
AAIP1 NNMP3 Sr
AAIP1 NNMP3 BNOTAG
AAIP1 NNMP3 L3
AAIP1 NNMP3 NNOTAG
AAIP1 NNMP3 C4
Z:--- NNFS7 BVc-S-
Z:--- NNFS7 BAGFP2
Z:--- NNFS7 BDg---
Z:--- NNFS7 BRV--2
Z:--- NNFS7 BDb---
Z:--- NNFS7 BAAFS7
Z:--- NNFS7 NNOTAG
```

The first two elements of each row are the vertex values f the dependency always in the order father, son. The third element determines the condition under which the indicator function is true:

- S, for *simple* type, has two values: 'l' or 'r'. It is true if the dependency is left or right, respectively.
- B, for *is between* feature, which is true if the vertex value immediately after the B sign is between the dependency values on the sentence surface.
- C, for *concord* feature, with three allowed values, value 3, value 4, and value 5. C3 is true if the dependency vertex values have their third position identical, C4 is true if the dependency vertex values have their fourth position identical, and analogously for C5.
- N, for *is not between* feature, which is true if the vertex value immediately after the N sign is not between the dependency values on the sentence surface.
- L, for *distance* feature type, which is true if the distance is less or equal to the number value immediately after the L sign.

It is up to the learning designer whether or not differences will be made[28] (therefore different feature instances) if:

- for the S feature, if the dependency values repeat themselves on the surface within the dependency or not. E.g., the feature

```
NNIXX AAMP2 Sl
```

[28] The same differences can be encountered for the procedure from Chapter 6.

could have four instances. If the following would be the surface situation

AAMP2 **X** NNIXX

the four different cases occur if:

- **X** does not contain AAMP2 neither NNIXX.
- **X** contains AAMP2 and it does not contain NNIXX.
- **X** does not contain AAMP2 and it contains NNIXX.
- **X** contains both AAMP2 and NNIXX.

In all of these situations it should be decided whether the indicator function is true (1) or false (0).

- for the B feature, if the between vertex value occurs on the surface between the dependency vertices once, or more than once

Explained on an example, if the between value is, e.g. B, the following cases could be distinguished for AAMP2 **X** B **Y** NNIXX:

- **X** contains NNIXX and does not contain AAMP2, while **Y** contains AAMP2 and does not contain NNIXX
- **X** contains AAMP2 and does not contain NNIXX, while **Y** contains NNIXX and does not contain AAMP2
- **X** does not contains AAMP2 neither NNIXX, and **Y** does not contain NNIXX neither AAMP2
- **X** contains both NNIXX and AAMP2, and **Y** contains as well both NNIXX and AAMP2

in the similar manner, the other feature variants can be easily deduced

- the N feature is similar to the B feature, with opposite meaning.

Other feature variants can be obtained if, e.g. between feature is tested only when the dependency distance is less than a pre-set value. While the other modifications helped the learning process, the distance limitation in this case did not positively influence the success rate. Therefore, the distance limitation can be used if we want to reduce the number of features in the feature vector, but it plays no significant role in including explicitly its (distance) value (the Distance type of rule).

The perceptron algorithm `perceptron_learning` is much faster than the `graphrule` and the feature vector can contain hundreds of thousands of features (indicator functions). The feature set can be extended also to features of various other types, as such containing lexical units (e.g., graph vertices of the m5l type).

Finding good set of features is currently the most important task for successful application of the perceptron algorithm and this problem is not solved yet.

7.5.1. Tests with additional feature types

Additional types of features have also been considered. To explain them better I will use the following abstract sentence notation, a string of tags:

$$t_1\ t_2\ ....\ t_{i-2}\ t_{i-1}\ \mathbf{t_i}\ t_{i+1}\ t_{i+2}\ ...\ t_{j-2}\ t_{j-1}\ \mathbf{t_j}\ t_{j+1}\ t_{j+2}\ ...\ t_{n-2}\ t_{n-1}\ t_n$$

Let the bolded tags be the dependency tags on which the feature is to be applied.

1. *Picture types* of features

   In the logic of picture segments as presented in Section 5.4 of Chapter 5 and assuming that t_i and t_j are the dependency nodes the first of which is the father (for the cases where the second is the father, their order in the feature description is swapped), the *before*, *between* and *after* types of features were tested.

   The *before* type of feature is of the form $\boldsymbol{t_i}\ \boldsymbol{t_j}\ t_k$, where k=1...(i-1).

   The *after* type of feature is of the form $\boldsymbol{t_i}\ \boldsymbol{t_j}\ t_k$, where k=(j+1)...n.

   The *between* type of feature is of the form $\boldsymbol{t_i}\ \boldsymbol{t_j}\ t_k$, where k=(i+1)...(j-1), which is identical to the "is between" (B) type of feature as described earlier in the text.

2. *Bigram and trigram types* of features

   The *bigram* type of features is of the following type:

   $\boldsymbol{t_i}\ \boldsymbol{t_j}\ t_{i-1}$ (if any, with respect to $\boldsymbol{t_i}$) and/or $\boldsymbol{t_i}\ \boldsymbol{t_j}\ t_{j-1}$ (if any, with respect to $\boldsymbol{t_j}$).

   The *trigram* type of features is of the following type:

   $\boldsymbol{t_i}\ \boldsymbol{t_j}\ t_{i-1}\ t_{i-2}$ (if any, with respect to $\boldsymbol{t_i}$) and/or $\boldsymbol{t_i}\ \boldsymbol{t_j}\ t_{j-1}\ t_{j-2}$ (if any, with respect to $\boldsymbol{t_j}$).

7.6. Results

Tenths of experiments were performed on various train data sets with the features as mentioned above and:

- paying attention to the repetition of nodes for the "simple" type
- without paying attention to repetition of nodes for the "is between" and "is not between" type.
- including the picture feature types
- including the bigram and trigram feature types.

Just to illustrate the length of the feature vector, such set of feature types results in approx. 10,000 feature instances automatically created from a set of 50 sentences.

The methodology of testing the significance of certain feature type has been the following:

1. Take a set of sentences
2. Create automatically all possible features as described above. Let the resulting set be *Features*.
3. Test the success of learning.
4. Select a single feature type end exclude it from *Features*.
5. Test the success of learning.

If the success from point 5 is the same (or only slightly different) compared to the success from point 3, the excluded feature can be omitted and is declared as non significant.

Features types found as non significant are the following:

- simple distance feature (as in Section 6.2.3),
- bigram features, and
- trigram features.

Those were therefore omitted in the larger experiment of perceptron-based learning and testing.

The picture types of rules helped in improving the performance.

The PDT graph from which the sentence graphs were derived has been of two types:

- the m5 type, i.e. a vertex value equals to the first five positions of the morphemic tag. The only exception has been the Z (punctuation) tag the second position of which was substituted by the value of the word-form for that tag.
- the m5l type, i.e. the vertex value equals to the first five positions of the morphemic tag including the previous modification of the punctuation tag, except for verbs, prepositions and conjunctions. In these cases the value was a compound one, equal to the concatenation of the m5 tag value and the lemma.

The latter was observed to be more successful.

The best results registered for the perceptron algorithm were lower than expected but still they are the highest results achieved so far:

- maximum of 78% while learning
- maximum of 72% while testing.

There was no need of more than 60 iterations. The main learning steps occur during the first few, approx. 5 to 15, iterations.

The perceptron algorithm does not significantly benefit from the dependency frequencies as described in Chapter 5. This is due to the large number of features, thus the benefits caused by the distance are well captured by other features as those of the "simple" type sensitive to repetition of nodes.

The zero initial vector of parameters causes that the initial tree is a random one. A success of such randomly chosen tree is most often between 18% and 30%. Assuming this, in terms of improvement the perceptron algorithm is the best of all algorithms mentioned so far (approx. 50% improvement).

The perceptron algorithm can be easily lexicalized - this is an important characteristic and one of the few possible directions for success rate improvement.

The fact that the perceptron algorithm uses special weight for every feature indicates that, the key to a more successful perceptron-based parsing lies in the feature types themselves and not in other parts of the algorithm.

7.7. Considerations for better features and algorithm performance

An important question remains unanswered: Which are the good features? Many features were presented but the performance did not significantly increase. A strict look at the data would say that the perceptron-based approach did not perform significantly better than the picture-based approach. But there is a significant 'but': compared to the other approaches, the perceptron algorithm offers big generality and a wide range of possibilities to consider, mainly in feature type definitions.

There is a certain 'hunger' for good features. What has been done in this work supports the fact that it will not be trivial to find them. The dependency length are long (Table 2.3), so trigram do not help. The information is very diverse as it can be noted from the picture evaluations (large proportion of zeros after combining negative and positive pictures).

The features as in Section 7.5 are perfectly suitable for single sentences but they perform worse on a set of sentences. E.g., if almost any sentence is isolated and we try to learn on it, we would get 100% success of learning (better, mimicking), as for the randomly selected Czech sentence (PDT file c101.am, #10) "Navíc současně vznikne písemný doklad, lze přenášet obrázky, grafy, fotografie, rukou psané texty apod.", or for the sentence (PDT file c101.am, #12) "Protože smyslem článku je napomoci orientovat se vám při výběru zařízení pro vaší potřebu, je nutné poněkud obšírněji vysvětlit, proč je nutné tento parametr uváděný v prospektech, brát jen jako orientační údaj.", or for many, many others. If both of the sentences are put together, the learning success is 79%, that is almost the same as for the larger training set (up to 78%) and much less than the 100% for the single sentence trainings. Why is that so?

Testing with learning on sets of single verb sentences which length is less than 15, the learning has been more successful and achieves 89%. The perceptron-based algorithm is sensitive to the sentence length. With respect to the above results, the sensitivity on sentence length is not for the reason of sentence brevity, but for the reason of grammatical diversity: shorter sentences are probabilistically simpler and the feature vector components 'do not beat each other'.

This work gives the following contributions towards answering the above stated questions. From our experiments it follows that the features are not consistently bound into the sentence context with respect to the language grammar - the same feature can have a positive contribution for some sentences, while a negative one for other sentences. The positive and negative 'bouncing' or 'beating' causes that the learning process does not grow beyond the stated 78%. There are two directions to solve this:

a. creating features with more sophisticated conditions,
b. including mutual effects and bindings on feature vector components.

Ad a., more sophisticated conditions would mean conditions on more input elements as, e.g. a whole dependency picture (which would yield enormous number of features that we would not be able to capture efficiently; moreover

such set would be very sparse in the sense of its statistical distribution), or manually created sets of features such that will code the (main) grammatical characteristics.

Ad b., preserving a simple set of features as in this work, but conditioning them mutually. This could be represented by a set of alternative feature vectors (assigned to each dependency) and a set of algebraic rules defined on them.

The current feature types can only operate on the surface sentence structure and cannot include conditions involving actual tree structure. There is additional research to be done on modifications of the sentence graphs adding, e.g., dummy edges, and consequently, accompanied by addition of a post MST phase for best tree deduction.

7.8. The percentages of the perceptron-based approach

The perceptron-based approach allows testing of various features rather efficiently. As also stated in the previous section our main topic of interest has been focused on feature testing. To be able to compare the method to the others I have restricted myself to m5 type of nodes or m5l type of nodes. For the case of m5l type of nodes I did not employ features that deal only with the lemmas, but I have treated the m5l type of nodes as a whole. The meaning of the m5 and m5l type of nodes is as in Chapter 2 and Chapter 5.

The training and testing were performed on parts of `anal_general_train` data from PDT. These parts were obtained from `l10*am` (total 9 files) and `c10*am` (total 9 files) files. Each PDT file consists of approx. 50 sentences. Sets of sentences from `c10*am` for training and from `l10*am` for testing were randomly selected - that is the way the number of sentences in Table 7.1 was taken. Table 7.1 summarizes the major tests performed in this chapter; to preserve the transparency of results the smaller experiments used for feature type selections are not included. If not stated otherwise the sentence graphs were on m5l type of nodes. The testing was performed using the gamma-parameters. The alpha parameters have also been tested - their success was approx. up to 5% less than the one using the gamma-parameters. The features used are as described in Section 7.5 excluding the non-significant ones as stated in Subsection 7.5.1.

Algorithm	Size and identification of train data	Size and identification of test data	Dependency accuracy	Note
The perceptron-based approach, Chapter 7	-	-	18%-30%	Initial success on random trees.
	100 sentences	on the train data	68%-78%	Range on various cross-validation experiments.
	400 sentences	100 sentences	67%-72%	Range on various cross-validation experiments.
	1 sentence	on the train data	100%	This is characteristic only for this approach and does not happen within the other approaches. Experiments confirmed the result on tenths of various sentences.
	100 sentences	on the train data	89%	Sentences with 1 verb and shorted than 15 verbs; only on m5 type of nodes.
	100 sentences	on the train data	64%-71%	Cross-validation on m5 type of nodes.
	100 sentences	on the train data	56%-64%	Cross-validation on m5 type of nodes and without the *is not between* feature and without picture type of features.

Table 7.1: The percentages of the perceptron-based approach

8. Get Almost Two For the Price of One

In this last chapter I would like to set up an experiment that occurred to me while finalizing this work, a dependency parsing-by-tagging experiment.

Let there be a tagger and a treebank. Then the following is possible.

Input: Treebank training set of pairs $(w_{[1:n]}, t_{[1:n]})$ and of pairs $(w_{[1:n]}, g_{[1:n]})$, where g = gov(t). $(w_{[1:n]}, g_{[1:n]})$ means that each sentence word form is assigned the morphemic tag of its governor node.

Example:

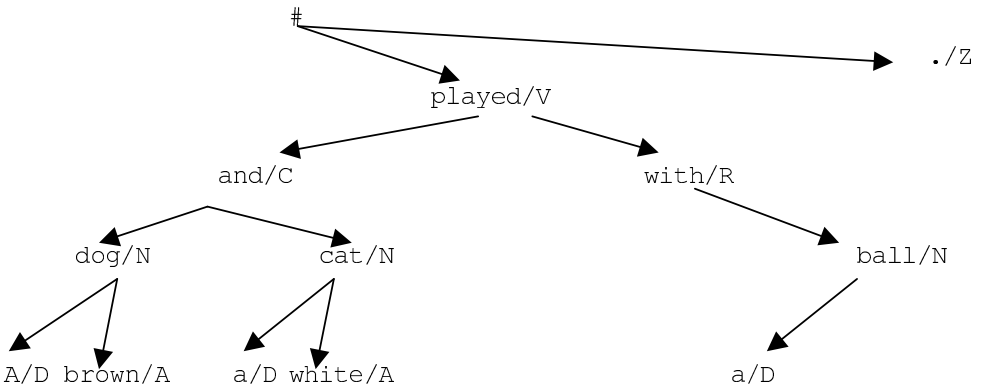
Let the following sentence

A brown dog and a white cat played with a ball.

is *m-tagged*[29] as follows

A/D brown/A dog/N and/C a/D white/A cat/N played/V with/R a/D ball/N ./Z

Let the following be its dependency tree.



Based on the treebank tree the tagging by governors train set examples can be generated. In our case, the example sentence would be *g-tagged*[30] as follows:

A/N brown/N dog/C and/V a/N white/N cat/C played/# with/V a/N ball/R ./#

Algorithm:

1. Train the tagger on m-tags.
2. Train the tagger on g-tags; in order to increase the g-tagging success rate this learning (and later on application) phase can be serialized to the previous one restricting the set of possible tags to those found within each sentence in the previous step. In our case, the aim would be to g-tag the sentence with the {D, A, N, V, C} tags.

---

[29] Tagged by morphemic tags (classical tagging).

[30] Tagged by morphemic tags, but such that belong to the governor node morphemic tag (*g-tag*).

3. Align the two outputs and resolve ambiguities.

Output: Analytical tree of the input sentence.

I will use an example in order to explain the alignment.

According to Figure 8.1, the first row is the row of m-tags (output of step 1), while the bottom one is the row of g-tags (output of step 2). If a tag does not repeat in the m-tag row the solution is straightforward. For our example (which includes also ambiguities) the straightforward nodes (dependencies) are shown in Figure 8.1.

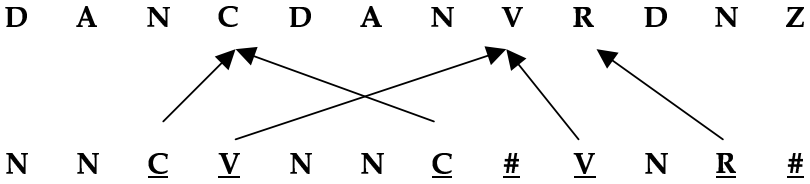


Figure 8.1: Alignment without a need of disambiguation

Figure 8.2 demonstrates the alignment to resolve the remaining dependencies.

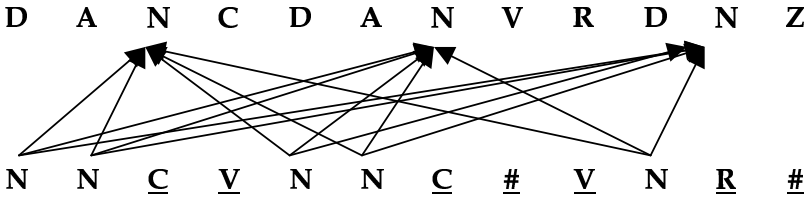


Figure 8.2: Alignment with a need of disambiguation

The simplest way (and luckily its works out rather well) is to use a shortest distance criterion, which means that the relatively closest (on the surface) element is selected, excluding zero distance since a node cannot be dependent to itself.

Applying a shortest distance criterion we obtain:

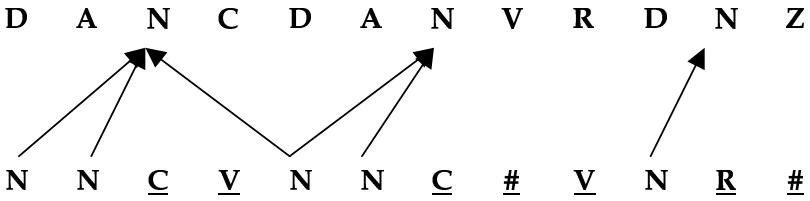


Figure 8.3: The shortest distance criterion

The situation in Figure 8.3 contains dependencies, which have not been resolved by the shortest distance criterion, since both dependency candidates have the same distance value. In this case one could use other criteria as: decision according to frequency values of left and right dependencies, or in case of adjective to noun attachments recognize accumulation of concorded adjectives, or in general test of grammatical concordance. For resolution of ambiguities of

alignment it is possible to employ also more sophisticated methods. The less successful the tagger is, the more important is the precision of the alignment algorithm.

For our case a frequency-based choice would prefer the left to the right dependency between an adjective and a noun, therefore the solution would be

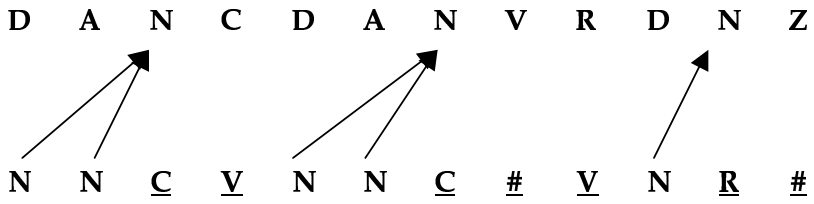


Figure 8.4: The frequency-based choice

The process demonstrated (Figure 8.1 to Figure 8.4) outputs an analytical tree of the input sentence.

I am deeply aware that a successful implementation would not be so couleur de rose. But there is enough background to expect that it would be rose enough to enter proudly in the category of 80% and above parsers. This expectation is based on past experience as in (Ribarov 2000a) or (Holan 2003) and as well as on the following facts from alignment tests on PDT and the assumptions afterwards.

Based only on the shortest distance criterion (if for the case of equal distance no frequency test is performed, but simply the left dependency is selected) and assuming that the taggers tag correctly, the following alignment success rates are found[31]:

- for the full tag alignment: 95.23%
- for the m5 (the first five positions) tag alignment: 94.88%
- for the m2 (the first two positions) tag alignment: 87%
- for the full tag without case category alignment: 93.38%

Assuming that tagging on analytical functions achieves 72% with a relatively bad tagger[32] we might expect that the 72%[33] can be significantly improved if the best current tagger is used. The structure of the analytical function tagset is the most similar one to the g-tags, therefore it is expected that the success rates could be similar. Moreover, in this, parsing-by-tagging

---

[31] The presented percentages represent dependency accuracy, that is a number of correctly found dependencies of a syntactic tree, which is the result of the alignment process. The percentages were calculated on 44 files (exactly all `c1*am` files) from `anal_general_training` set of PDT, which is more than 2000 sentences. According to the way the dependencies are obtained, the percentage rates the lowest possible.

[32] The rule-based tagger designed originally for English. The claim 'relatively bad' is for tagging of Czech when compared to the current best tagger with approx. 95% success rate on the full tagset.

[33] The 72% tagging accuracy on analytical functions (Figure 3.11) has been obtained with the rule-based tagger that achieves almost 80% on tagging of Czech on its full tagset (Table 3.2).

experiment, the tagset of the sentence to tag in step 2 can be reduced (intersected) by the tagset formed from the tags from step 1 (as stated earlier). Such reduction will, as shown in (Hladká, Ribarov 1998), improve the tagging accuracy. The alignment percentages can also be improved - the presented ones are the lowest possible. It is to test, whether omitting the case (the source of most tagging errors) would be positively compensated with the slightly lower alignment performance.

This approach demonstrates that it is approximately possible to transform the parsing problem to a tagging one. This would place the tagger as a central NLP device.

Common ground for the tagger and the parser would be a creation of good features as a prime direction to follow.

Bearing in mind that the best current taggers are statistical ones, I cannot conclude differently than, in order to perform better, statistics needs rules (e.g. having in mind the alignment), and that rules need statistics (having in mind the best current taggers). Therefore, improvement of the success rates of our taggers, parsers, aligners has a common denominator.

9. Final Notes

From the time perspective, this work presents an eight year passage starting in 1996. At that time various modifications of the rule-based approach for dependency parsing of Czech were tested. The work done and also published as in (Hajič, Ribarov 1997) had rather positive reply, which has been a motivation to continue and complete the work in the years afterwards (Chapter 3). The second rule modifications were completed back in 1998 and 1999. At that time, together with my colleague Barbora Hladká, we decided to understand closer the effect of tagset size on the learning success rates. Our trials have been summarized in (Hladká, Ribarov 1998). The 1999 and 2000 have been years when a 'searching' has started. A rather in-the-dark searching - there was not enough evidence on how to make the rules perform better. It has been a search for ideas.

In the trials to understand better the structure of the information from the corpus (later on there were already 'alpha' and 'beta' versions of the Prague Dependency Treebank available) experiments with non-linear and chaotic measures were employed. Even without including syntactic information but studying a morphological context, the measures (some of which are presented in (Ribarov 2000b)) indicated that the 'relevant things' are most likely within an environment of 15 words and that information outside of such region would not improve significantly the learning tools. Nowadays, with the PDT available, we have, as demonstrated in Chapter 2, verified that the average dependency length is 11, which, together with the dependency elements themselves and their close vicinity falls nicely in the 15-word region as predicted by the non-deterministic measures. Compared to small contexts as those of trigrams, this has indicated that there is a big problem to overcome, a problem which has still no efficient algorithmic answer.

In the meanwhile a very successful parsing of Czech has been done as published in (Collins 1999) - this significant work together with the Charniak's experiment (Charniak 2001) formed a high standard line to cross. Nevertheless also these work in terms of their offer of possible improvements lack what we all lack - new ideas. Almost all current tendencies are streamed as a search for new (not born until now) features.

It is known that by lexicalizing the features better success rates can be obtained. That is what Collins and Charniak did, this is what is also obvious from the more successful training on m5l sentence graph nodes than on m5 sentence graph nodes (Chapter 7). I was aware that if I had focused more on the lexicalization I would have got better results in terms of dependency accuracy. The perceptron-based approach from Chapter 7 would have no problems with lexicalization. The reasons why I did not lexicalize are:

- In order to compare the perceptron-based approach to the previous work, I have used similar information, therefore mainly morphemic tags.
- The lexicalization performed in (Collins et al. 1999) deals with many features, but all of them are of the same type (structure) as those dealing only with morphemic tags. There is a dependency rate improvement, in fact, it is the

probably best such improvement for such types of features. But, I believe that with qualitatively better features it is possible to achieve more than that. This explains why I did not, for the purpose of development of this work, find attractive the lexicalization of the features - extending them with lexicalizations would not bring qualitatively new result that we have already seen in the work of Collins or Charniak.

- For the purpose of parsing tool development I would definitely use lexicalized features; what I have written above does not deny their significant contribution.

The experiment as described in Section 3.8 of Chapter 3 would like to help to overcome the narrow trigram margins. The PDT nice saturation figures in Chapter 2, gave rise to the idea that the solution could be restricted by the PDT itself. Thus, I have decided to work only with such dependencies which exist in PDT and are defined as a dependency between two morphemic tags[34] (counting with lexical units would not saturate the treebank graph).

The graph theory (including theory of matroids) offers nice formalisms for dealing with abstract structures. It occurred to me that a maximum spanning tree could be the true tree of a sentence which graph edges are 'somehow' weighted. The RBA approach has been the closest one and the one I had a good experience with, so I have tried to reformulate it such as to be able to work on graphs. This phase is included in Chapter 6. I though of the process as a process of game where the weights represent dependency 'lives' - before a dependency 'dies' it should be repeatedly canceled by several constraints (rules). Taking the MST at the end of such process should results in a decent dependency tree, I believed.

Searching for the maximum spanning tree algorithm has not been direct. I have not found any graph theory book where this is described significantly well. After internet searches I have found the page of Shanchieh Jay Yang which had the answer I have searched for. At that time I had a small group of enthusiastic students at the Czech Technical University and we have decided to examine the efficiency of the algorithm. Consequently, the first version of the MST algorithm has been written in Perl. We have verified that the algorithm is fast enough and suitable to be applied on sentence graphs; there was no need of employing methods in order to determine the tree root, since PDT trees have an artificial root sign which always represents the root. After refining it, I have used the MST in a series of experiments I have performed from then on.

The application of the MST algorithm for the modification of the perceptron algorithm has not been immediate. The results presented in Chapter 7 result from my step by step approaches to the new shape of the perceptron dependency parsing.

Additional new idea has been the one as defined in Chapter 8, the parsing-by-tagging problem. Something I will most likely run into very soon.

In order to achieve higher success rate I did not want to start combining various approaches just for the purpose of increasing the success rate. It would

[34] This does not limit us in dealing with lexicalized features or rules.

have made no sense to run after percentage improvements if in fact the methods had performed under the 70% limit. I wanted to increase those 70% only by the methods themselves or by new ones. Therefore, it seemed more sensible to me to devote to the algorithms and to reveal their functionality. Undoubtedly, with the help of many pre- or post-processing hints as word groupings, idiom and phrase fixing, help of valency dictionary or other kind of dictionary, separation of short and non verbal sentences, parenthetical parts, enumeration lists etc., the success would rise as there are many such elements in PDT. Such modifications were deliberately not done here. Thus, from its purely algorithmic perspective, the presented results are original and, at least within the Czech environment, best of their kind. The methods worked out and presented here are easy to re-use, to verify and to improve. This work also presents many aspects relevant for possible hybrid approaches and super parsing of Czech.

Let me summarize that for the purpose of automatic building of a dependency tree this work has presented the following new elements in dependency parsing:

- the rule-based approach accompanied by a variety of modifications,
- the perceptron-based approach and its modifications,
- two relatively well-performing naive parsers,
- a definition of a parsing-by-tagging experiment.

Every chapter of this work ends with concluding remarks on the discussed topic and all main experiment results are gathered in the Appendix. I will therefore not repeat myself here. Yet, some additional notes follow.

The performed experiments and achieved results for the rule-based parsing as in Chapter 1 and Chapter 3 do not give future prospects for their application the way they currently are. Anyway, if one wants to follow this direction, it is advisable that the problem is divided in two stages - a stage of "word grouping" or chunking such that dependent words will get reasonably close together and afterwards, a stage of local dependency tree building. The rule-based approach, especially its graph variant (Chapter 6) learns well on sentences, which share some common characteristics or are of shorter length. The rule-based approaches share their advantage of being capable of learning from a relatively small training set. Nevertheless, this is of no practical use in our case, since the absolute success rates are low. The only exception is the rule-based graph variant, which demonstrated to perform at least as well as the picture-based naive parser and at the same time indicates space for further improvement.

If I were to point out what I find most challenging it would be the use of the rooted directed maximal spanning tree - this idea allowed me to perform the modification of the perceptron algorithm, to play with the sentence graph in the rule-based way, and to play with frequencies and pictures. Its low complexity allows the MST operation to be performed repeatedly without significant delays of process execution time.

I was flattered by how well, without any training the MST performed on dependency frequencies. It is a fact that the first computational experiments on parsing of Czech did not exceed 40% of success rate, while from current perspective something as the frequency, without any learning, significantly outperforms our algorithmic ancestors.

While playing with the pictures, it was nice to notice that the picture approach does not depend on sentence length, that is not usually the case with learning algorithms, e.g. the rule-based one. The fact that the pictures perform non-neglectingly well indicates that there is something, yet to be found, out there between the father and the son node or beyond. The fact that pictures do not perform better than they do, indicates that the difficult stuff to do with Czech syntax starts above the 70%.

I was pleasantly surprised by how fast the perceptron algorithm converges and by how many features it can support. This method deserves big attention in the direction of parsing and in the direction of tagging. Additional good news about the perceptron is in the possibility to incorporate any type of features, which deal with the sentence surface elements, including manually designed features.

The à la RBA approach, when trained on small sets extracted the sentence dependencies with a success of approx. 85% (as stated in Chapter 6, the application afterwards has been unfortunately much worse). A way how to make this local character a global one might be in combining the rules and the perceptron by taking `graphrule` best rules (from a series of à la RBA trainings on smaller data sets) as perceptron features. As shown in Chapter 7 the rules and the features can share the same form.

The parsing-by-tagging idea as presented in Chapter 8 demonstrates that it is possible to transform the parsing problem to a tagging one approximately. This would place the tagger as a central NLP device and extend it to a tool for one-to-one assignment of linguistic categories to words. Common ground for the tagger and the parser would be a creation of good features as a prime direction to follow.

Let me conclude in the same way as in Chapter 8, but more generally that, in order to perform better, statistic approaches need rules and, rule-based approaches need statistics to perform better with their rules.

I hope that the results presented here will be of benefit to my colleagues who are primarily interested in the performance of the learning procedures. I have written this work as an inspiration and as a possibility to contribute to the NLP community with some new input and fresh ideas. And, not only from good results, but also by learning from our mistakes, we can get better.

References

Bémová, A. et al. (1997). "Anotace na analytické rovině: návod pro anotátory", Technical Report of the Institute of Formal and Applied Linguistics TR-1997-03, Faculty of Mathematics and Physics, Charles University, Prague. English version available as a part of PDT 1.0 documentation.

Böhmová, A. (2001). "Automatic Procedures in Tectogrammatical Tagging", The Prague Bulletin of Mathematical Linguistics vol. 76, Charles University Press.

Bock, F. (1971). "An algorithm to construct a minimum spanning tree in a directed network", Developments in Operations Research, Gordon and Breach, NY, pp. 29-44.

Brill, E. (1992). "A simple rule-based part of speech tagger", Proceedings of the Third Conference of Applied Natural Language Processing, ACL, Trento, Italy.

Brill, E. (1993a). "Automatic grammar induction and parsing free text: A Transformational-Based Approach", In the Proceedings of the 31st Meeting of the Association of Computational Linguistics in Columbus, Ohio, and as well in the Proceedings of the ARPA Human Language Technology Workshop in Princeton, N.J.

Brill, E. (1993b). "Transformation-based error-driven parsing", Proceedings of the Third International Workshop on Parsing Technologies, Tilburg, The Netherlands.

Brill, E. (1993c). "A Corpus-Based Approach to Language Learning", A Dissertation in the Department of Computer and Information Science, University of Pennsylvania.

Camerini, P.M, Fratta, L., Maffioli, F. (1979). "A note on finding optimum branchings", Networks vol. 9, pp. 309-312.

Charniak, E. (2001), *Internal studies and inter-institutional communication*.

Chu, Y.I. and Liu, T.H. (1965). "On the shortest arborescence of a directed graph", Science Sinica vol 14, pp. 1396-1400.

Collins, M. (2002). "Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms", In Proceedings of EMNLP 2002.

Collins, M., Hajič, J., Ramshaw, L., Tillmann, C. (1999). "A Statistical Parser for Czech", Proceedings of ACL´99, Maryland, USA.

Edmonds, J. (1967). "Optimum branching", Research of the National Bureau of Standards, 71B, pp. 233-240.

Elworthy, D. (1995). "Tagset Design and Inflected Languages", Proceedings of the ACL Sigdat Workshop, Dublin, Ireland, pp. 1-9.

Hajič, J., Ribarov, K. (1997): "Rule Based Dependencies", Workshop on Empirical Learning of NLP Tasks, 9th European Conference on Machine Learning, Prague.

Hajič, J. (1998). "Building a Syntactically Annotated Corpus: The Prague Dependency Treebank", In Issues of Valency and Meaning - Studies in Honour of Jarmila Panevová, edited by Eva Hajičová, Karolinum, Charles University Press, Prague, pp. 106-132.

Hajič, J. (2004). Disambiguation of Rich Inflection (Computational Morphology of Czech), Faculty of Mathematics and Physics, Charles University, Prague.

Hajič, J., Vidová-Hladká, B., Böhmová, A., Hajičová, E. (2003). "The Prague Dependency Treebank: A Three-Level Annotation Scenario", In Building and Using Syntactically Annotated Corpora, edited by Anne Abeille, Kluwer Academic Publishers.

Hajičová, E. editor (1995). "Text and Inference Based Approach to Question Answering", Theoretical and Computational Linguistics vol. 3, Faculty of Philosophy, Charles University, Prague.

Hajičová, E., Sgall, P. (1980). "A Dependency Based Specification of Topic and Focus", Statistical Methods in Linguistics (SMIL) 1-2, pp. 93-140.

Hladká, B., Ribarov, K. (1998). "Part of Speech Tags for Automatic Tagging and Syntactic Structures", In Issues of Valency and Meaning - Studies in Honour of Jarmila Panevová, edited by Eva Hajicová, Karolinum, Charles University Press, Prague, pp. 226-240.

Holan, T. (2003). "K syntaktické analýze českých (!) vět", Proceedings of MIS2003, Matfyzpress, Prague, pp. 66-74.

Holan, T. (2004). "Tvorba závislostního syntaktického analyzátoru", Proceedings of MIS2004, in print.

Horák, A. (2000). Analýza znalosti ve větě, PhD Thesis at the Faculty of Informatics at the Masaryk's University in Brno.

Kirschner, Z. (1987). "APAC3-2: An English-to-Czech Machine Translation System", Explizite Beschreibung der Sprache und automatische Textbearbeitung XIII, Faculty of Mathematics and Physics, Charles University, Prague.

Kirschner, Z. (1988). "A Dependency-Based Analysis of English for the Purpose of Machine Translation", Explizite Beschreibung der Sprache und automatische Textbearbeitung IX, reprint (2nd edition), Faculty of Mathematics and Physics, Charles University, Prague.

Klavans, J.L., Resnik, P. - editors (1996). The Balancing Act. The MIT Press.

Králíková, K., Panevová, J. (1990). "ASIMUT - A Method for Automatic Information Retrieval from Full Texts", Explizite Beschreibung der Sprache und automatische Textbearbeitung XVII, Faculty of Mathematics and Physics, Charles University, Prague.

Kuboň, V. (2001). Problems of Robust Parsing of Czech, PhD Thesis at the Faculty of Mathematics and Physics, Charles University, Prague.

Matoušek, J., Nešetřil, J. (2000). Kapitoly z diskrétní matematiky. Karolínum Press, Prague.

Mírovský, J. (NetGraph). PDT search engine programmed by Jiří Mírovský, Faculty of Mathematics and Physics, Charles University, last release at: http://quest.ms.mff.cuni.cz/netgraph.

Mitchell, T.M. (1997). Machine Learning, McGraw Hill Companies, Inc.

Lawler, E. (1976). "Combinatorial optimization: networks and matroids", Saunders College Publishing.

Oliva, K. (1989). "A Parser for Czech Implemented in Systems Q", Explizite Beschreibung der Sprache und automatische Textbearbeitung XVI, Faculty of Mathematics and Physics, Charles University, Prague.

PDT 1.0: The Prague Dependency Treebank, PDT 1.0, has been developed by the Institute of Formal and Applied Linguistics and the Center for Computational Linguistics, see http://ufal.mff.cuni.cz/ and/or http://ckl.mff.cuni.cz/.

Petkevič, V. (1995). "A new formal specification of underlying structures", Theoretical Linguistics vol. 21, pp. 7-61.

Pajas, P. (TrEd). Tree Editor programmed by Petr Pajas, Faculty of Mathematics and Physics, Charles University, last release at: http://ckl.ms.mff.cuni.cz/~pajas/tred.

Panevová, J., Böhmová, A., Hajičová, E., Sgall, P., Ceplová, M., Řezníčková, V. (2000). "A Manual for Tectogrammatic Tagging of the Prague Dependency Treenbank", English version in Technical Report of the Institute of Formal and Applied Linguistics and the Center for Computational Lingustics TR-2000-09, Czech version in Technical Report of the Institute of Formal and Applied Linguistics TR-1999-07, Faculty of Mathematics and Physics, Charles University, Prague.

Plátek, M., Sgall, J., Sgall, P. (1984). "A Depndency Base for a Linguistic Description", In Contributions to Functional Syntax, Semantics, and Language Comprehension edited by Petr Sgall, Academia, pp. 63-97.

Ratnaparkhi, A. (1996). "A maximum entropy part-of-speech tagger", Proceedings of the empirical methods in natural language processing conference.

Ribarov, K. (1996). Automatic natural language grammar generation, MSc Thesis at the Faculty of Mathematics and Physics, Charles University.

Ribarov, K. (2000a). "Rule-Based Tagging: Morphological Tagset versus Tagset of Analytical Functions", Proceedings of LREC'2000.

Ribarov, K. (2000b). "The (Un)Deterministic Nature of Morphological Context", Proceedings of LREC'2000.

Ribarov, K. (2002). "On Rule Based Parsing of Czech", The Prague Bulletin of Mathematical Linguistics vol. 77, Charles University Press.

Sgall, P., Hajičová, E., Panevová, J. (1986). The Meaning of the Sentence in Its Semantic and Pragmatic Aspects, edited by Jacob L. May, D. Reidel Publishing Company, Dordrecht. (Published as well by Academia, Prague, 1986).

Stráňáková Markéta (2001). Homonymie předložkových skupin a možnost jejich automatického zpracování, PhD Thesis at the Faculty of Mathematics and Physics, Charles University, Prague.

Tarjan, R.E. (1977). "Finding Optimum Branching", Networks vol. 7, pp.25-35.

Weisheitelová, J., Králíková, K., Sgall, P. (1982). "Morphemic Analysis of Czech", Explizite Beschreibung der Sprache und automatische Textbearbeitung VII, Faculty of Mathematics and Physics, Charles University, Prague.

Zeman, D. (1998). "A Statistical Approach to Parsing of Czech", The Prague Bulletin of Mathematical Linguistics vol. 69, Charles University Press.

Zeman, D. (2001). "Parsing with Regular Expressions: A Minute to Learn, a Lifetime to Master", The Prague Bulletin of Mathematical Linguistics vol. 75, Charles University Press.

Žáčková, E. (2002). Parciální syntaktická analýza (češtiny), PhD Thesis at the Faculty of Informatics at the Masaryk's University in Brno.

Appendix

This appendix gathers in Table A.1 and Table A.2 the percentage tables of this work, i.e. Table 3.8, Table 5.5, Table 6.1 and Table 7.1. The percentages from Chapter 8 are included as well. The original (partial) tables contain footnotes and explanations, which are omitted here.

Algorithm	Size and identification of train data	Size and identification of test data	Dependency accuracy	Note
RBA, Chapter 1, *With no treebank available all sentences were hand crafted only for the purpose of development of the algorithm.*	25 sentences	20 sentences	42%	
	100 sentences	*the same as above*	42%	
RBA, Chapter 3, *PDT was under development; the used sentences at that time consisted of many linguistic inconsistencies (cleaned several years afterwards). During that period the morphemic tagset has also been changed several times before it settled to the positional one.*	100 sentences	on the train data	47%	On $PosTag_{1171}$ as in Table 3.3.
	the same as above	on the train data	65%	On $PosTag_{34}$ as in Table 3.4. The same accuracy was observed on modified positional tagset, taking only the first two positions, i.e. POS and subPOS..
	100 sentences *[1]*	400 sentences	43%	On POS and subPOS only.
	200 sentences	*the same as above*	slightly less than 43%	On POS and subPOS only.
	300 sentence	*the same as above*	slightly less than 43%	On POS and subPOS only.
	100 sentence, *the same as [1]*	on the train data	73%	On POS and subPOS only, using error function as in Section 3.5.
	100 sentences with 1 verb	200 sentences with one verb	47%	On POS and subPOS only.
	100 sentences	200 sentences	56%	No morphemic but analytical function tags were used.
Frequency-based MST parser, Chapter 5, Section 5.3	`anal_general_train`	`anal_general_etest`	63.27%	Freq/m and m5, as in Table 5.1.
	the same as above	on the train data	63.96%	Freq/m and m5, as in Table 5.1.
	the same as above	`anal_general_etest`	58.46%	Freq/m and m5, and not considering dependency length bigger than 5.
	the same as above	*the same as above*	59.07%	Freq/m and m5, and not considering dependency length bigger than 7.

	the same as above	*the same as above*	60.23%	Freq/m and m5, and not considering dependency length bigger than 11, which is average dependency length.
	the same as above	*the same as above*	61.25%	Freq/m and m5, and not considering dependency length bigger than 15.
	the same as above	All sentences from `anal_general_e test with no verbs`	67.35%	Freq/m and m5.
	the same as above	All sentences from `anal_general_e test with 1 verb`	70.37%	Freq/m and m5.
	the same as above	All sentences from `anal_general_e test with 2 verbs`	62.02%	Freq/m and m5.
	the same as above	All sentences from `anal_general_e test with more than 4 verbs`	55.20%	Freq/m and m5.
	the same as above	All sentences from `anal_general_e test with one verb and at most 8 tokens`	84%	Freq/m and m5.
	the same as above	All sentences from `anal_general_e test with at most 8 tokens`	77%	Freq/m and m5.
Picture-based MST parser, Chapter 5, Section 5.4 *All pictures consist of m5 type of nodes (the first 5 positions of the morphemic tag), and no lexical units.*	`anal_ general_ train`	on the train data	68.34%	All segments and inverse distance.
	the same as above	*the same as above*	52.95%	All segments and without distance.
	the same as above	*the same as above*	52.95%	Only the *after* segment with inverse distance.
	the same as above	*the same as above*	51.41%	Only the *before* segment without distance.
	the same as above	*the same as above*	53.99%	Only the *in* segment with distance.
	the same as above	*the same as above*	58.06%	Only the *in* segment without distance.
Combination of the above two naive parsers	*the same as above*	*the same as above*	at least 70%	Freq/m and m5, and all segments and inverse distance

The table continues ...

À la RBA parsing, Chapter 6	-	-	63%	Initial structure dependency accuracy.
	100 sentences	on the train data	82%-87%	Interval of dependency accuracy on several cross-validation experiments.
	400 sentences	100 sentences	66%	
	the same as above	on the train data	69%	
	`c101.am` file	on the train data	lower than above	On m5l type of nodes.
The perceptron-based approach, Chapter 7	-	-	18%-30%	Initial success on random trees.
	100 sentences	on the train data	68%-78%	Range on various cross-validation experiments.
	400 sentences	100 sentences	67%-72%	Range on various cross-validation experiments.
	1 sentence	on the train data	100%	This is characteristic only for this approach and does not happen within the other approaches. Experiments confirmed the result on tenths of various sentences.
	100 sentences	on the train data	89%	Sentences with 1 verb and shorted than 15 verbs; only on m5 type of nodes.
	100 sentences	on the train data	64%-71%	Cross-validation on m5 type of nodes.
	100 sentences	on the train data	56%-64%	Cross-validation on m5 type of nodes and without the *is not between* feature and without picture type of features.
The parsing-by-tagging experiment, Chapter 8	0 sentence, no need to train	All `c1*am` files from `anal_general_training` part of PDT, i.e. over 2000 sentences	95.23%	On full morphemic tagset, assuming only closest distance candidates and in case of ambiguity preference of left dependency.
	the same as above	*the same as above*	94.88%	*the same as above*, but on m5 nodes.
	the same as above	*the same as above*	87%	*the same as above*, but on m2 nodes.
	the same as above	*the same as above*	93.38%	*the same as above*, but on full tagset and excluding the case category.

Table A.1: The percentages

Table A.2 is a table of improvements expressed via dependency accuracy. By improvement I denote the dependency accuracy percentage rate by which the algorithms have raised up their initial success (the success before their performance) on a test set, according to Table A.1: the initial tree structure (right

chain structure) for RBA, the initial sentence graph with frequencies and inverse distance for à la RBA, and the initial random trees on a sentence graph for the perceptron-based approach.

Algorithm	Improvement (approximate)
RBA, Chapter 1	+8.5%
RBA, Chapter 3	+16%
À la RBA	+3%
Perceptron-based	+50%

Table A.2: The improvements

I would like to note that in terms of further improvement, I find the study of the algorithms including the rules and the features very important. The here-presented percentages should be used only for better preparation of our future work.

Notes:

Notes: